

**OUTDOOR NAVIGATION OF MOBILE ROBOTS
BY USING ULTRASONIC SENSORS**

**M. Sc. Thesis by
Murat GÜNDOĞDU, B.Sc.**

Department : Inter-disciplinary

Programme: Mechatronics Engineering

FEBRUARY 2006

**OUTDOOR NAVIGATION OF MOBILE ROBOTS
BY USING ULTRASONIC SENSORS**

**M. Sc. Thesis by
Murat GÜNDÖĞDU, B.Sc.
518021003**

Date of submission : 19 December 2005

Date of defence examination: 30 January 2006

Supervisor (Chairman): Assoc. Prof. Dr. Hakan TEMELTAŞ

Members of the Examining Committee: Prof.Dr. Serhat ŞEKER

Prof.Dr. Ata MUGAN

FEBRUARY 2006

**DIŞ ORTAMDA ULTRASONİK SENSÖRLER İLE
MOBİL ROBOTUN HEDEFE GİTMESİ**

**YÜKSEK LİSANS TEZİ
Müh. Murat GÜNDOĞDU
518021003**

**Tezin Enstitüye Verildiği Tarih : 19 Aralık 2005
Tezin Savunulduğu Tarih : 30 Ocak 2006**

**Tez Danışmanı : Doç.Dr. Hakan Temeltaş
Diğer Jüri Üyeleri: Prof.Dr. Serhat Şeker
Prof.Dr. Ata Mugan**

ŞUBAT 2006

ACKNOWLEDGEMENTS

This study has been performed under supervision of Assoc. Prof. Dr. Hakan Temeltaş in Istanbul Technical University. I would like to thank to him for his very valuable guidance , suggestions, advice and encouragement at all stages of this work It was a great honor working with him.

I would like to express my sincere feelings to my family members for their encouragements during my study and also I would like to thank my family for their patience and support in everything all my life.

February, 2006

Murat GÜNDOĞDU

CONTENTS

ACKNOWLEDGEMENTS	iii
CONTENTS	iv
ABBREVIATIONS	v
TABLE LIST	vi
FIGURE LIST	vii
SYMBOL LIST	ix
ÖZET	x
SUMMARY	xi
1. INTRODUCTION	1
2. MODELLING OF OUTDOOR ENVIRONMENT	10
2.1. Regular cells	12
2.2. Quadtree cells	14
2.3. Framed-Quadtree cells	15
3. PATH PLANNING	20
3.1. The A* Algorithm	22
3.2. The D* Algorithm	38
4. SENSOR MODEL	42
5. SIMULATION STUDIES	50
5.1. Traverse Length	53
5.2. Memory Usage	53
5.3. Execution Time	54
6. CONCLUSION	56
REFERENCES	57
CURRICULUM VITAE	60

ABBREVIATIONS

LITA: Life in the Atacama

NE: North East

NW: North West

SE: South East

SW: South West

TABLE LIST

	<u>Page No</u>
Table 2.1 Different states for regular map calculation in Matlab program	13
Table 5.1 Summary of results	55

FIGURE LIST

		<u>Page No</u>
Figure 1.1	Mobile robot ASIMO by HONDA	1
Figure 1.2	A man searching for a mine	2
Figure 1.3	Hyperion in the Salar Grande, Atacama Desert, April 2003.	3
Figure 1.4	Functional diagram of the map building and navigation processes	5
Figure 2.1	Realmap generated by Matlab for simulation (64*64)	10
Figure 2.2	Outdoor modelling using regular cells	13
Figure 2.3	Quadtree tessellation of a region due to a single obstacle and the corresponding quadtree data structure.	14
Figure 2.4	An example of a path generated using quadtrees	15
Figure 2.5	An example of a path generated using framed-quadtrees	16
Figure 2.6	Data structure for framed quadtrees	17
Figure 2.7	Divide function for framed-quadtree structure	18
Figure 2.8	As the splitting of a framed-quadtree node occurs (the dotted rectangles show the first step of splitting), some links like edge gh are not affected, while others like edge ik are affected	19
Figure 3.1	Example direction matrice for A* algorithm with regular grids, starting position	23
Figure 3.2	Example real map for A* algorithm with regular grids, starting position	24
Figure 3.3	Example mobile robot map for A* algorithm with regular grids, starting position	24
Figure 3.4	Example direction matrice for A* algorithm with regular grids, location A	25
Figure 3.5	Example real map for A* algorithm with regular grids, location A	26
Figure 3.6	Example mobile robot map for A* algorithm with regular grids, starting location A	26
Figure 3.7	Example direction matrice for A* algorithm with regular grids, location B	27
Figure 3.8	Example real map for A* algorithm with regular grids, location B	28
Figure 3.9	Example mobile robot map for A* algorithm with regular grids, starting location B	28
Figure 3.10	Flow chart of matlab program main A* algorithm	29
Figure 3.11	Flow chart of matlab program direction_regular	30
Figure 3.12	Example mobile robot map for A* algorithm with framed quadtrees Starting position	31

Figure 3.13	Example mobile robot map for A* algorithm with framed quadtrees Location A	32
Figure 3.14	The direction and cost values generated by simulation program	33
Figure 3.15	Flow chart of main program for A* with framed quad tree cells	34
Figure 3.16	Flow chart of direction program for framed quad tree cells	35
Figure 3.17	Typical connection patterns of a framed-quadtree	36
Figure 3.18	Direction and cost matrices cells which are kepted same at location A* for D* algorithm	40
Figure 3.19	Direction and cost matrices updated at location A* for D* algorithm	41
Figure 4.1	Current model of the sonar sensor.	44
Figure 4.2	Objects in different positions can give the same distance reading by ultrasonic sensing	45
Figure 4.3	Specular Reflection: False reflections may occur for large angles of incidence by ultrasonic sensing	46
Figure 4.4	Sensor function flow chart	47
Figure 4.5	Realmap for sensor simulation	48
Figure 4.6	Robotmap after ultrasonic sensors has fired.	49
Figure 5.1	Realmap for simulation(32*32)	50
Figure 5.2	Realmap for simulation(64*64)	51
Figure 5.3	Realmap for simulation(96*96)	51
Figure 5.4	Ultrasonic sensor positions on mobile robot	52
Figure 5.5	Comparison of traversel lenth between regular grid and framed-quadtree mapping	53
Figure 5.6	Comparison of memory usage in between regular grid and framed-quadtree mapping	54
Figure 5.7	Exacution time comparison	55

SYMBOL LIST

S	: Start point
G	: Goal point
ρ	: Distance to the robot
θ	: The angle of the main axis of the sonar beam
β	: Beam aperture
γ	: Width of the region of uncertainty
d	: The range measurement returned by sensor
ϵ	: Empty probability density
ϕ	: Occupied probability density

ÖZET

DIŞ ORTAMDA ULTRASONİK SENSÖRLER İLE MOBİL ROBOTUN HEDEFE GİTMESİ

Murat GÜNDOĞDU

Bu çalışmanın amacı ultrasonik sensör kullanan mobil robotun dış ortamda bir yerden bir yere gitmesi için gerekli olan yol planlamasının grid, quadtree ve framed-quadtree yöntemleri ile oluşturulmuş haritalarda A* ve D* algoritmaları kullanılarak yapılmasıdır. Matlab programında yazılmış olan simülasyon programı ile gidilen yolun uzunluğu , zaman ve kullanılan hafıza açısından sonuçlar karşılaştırılmıştır. Simülasyon programı kullanılarak dış ortamlar için yol planlamasında dinamik A* algoritmasının ve büyük alanların haritalanmasında framed-quadtree yönteminin kullanılmasının hafıza, zaman ve yol uzunluğu açısından avantajları gösterilmiştir.

SUMMARY

OUTDOOR NAVIGATION OF MOBILE ROBOTS BY USING ULTRASONIC SENSORS

Murat GÜNDOĞDU

This work reports outdoor navigation of mobile robots equipped with ultrasonic sensors. A simulation environment is created with Matlab program and by using this simulation environment map building techniques (regular grids, quadtree mapping and framed-quad tree mapping) and path planning algorithms (A* and D *) are compared in traverse length, time and used memory. Simulation results shows how the use of framed-quadtrees leads to paths that are shorter and more direct than when other representations like regular grids and quadtrees are used. Combining an optimal path planning algorithm like D* with framed-quadtree map representation has the benefit of optimal path planning in traverselenth and time while minimizing the memory requirements.

1. INTRODUCTION

Mobile robots were originally designed as reprogrammable, multifunctional devices and to move material in industrial environments, the envisioned abilities of robotics systems has risen tremendously over the last few decades. Mobility, particularly, has been greatly improved, adding versatility to these systems. It was expected at the beginning of the nineties that in 2000 there will be 50000 independently operating autonomous service robots in the production areas and in the service sector [1]. Vehicles guided by a magnetic or optical line in industrial environments are some examples of robots in 2005. And in practice autonomously acting mobile service systems, for instance systems not restricted to follow a line are rarely used. One main example for mobile robots is ASIMO (Figure 1.1) which is recently present by Honda. There is a gap between prognoses and reality and this is because of the complexity of the environment in which the robots have to act.



Figure 1.1 : Mobile robot ASIMO by HONDA

Hazardous waste cleanup, spatial exploration, demining (figure 1.2) or similar applications in changing and unknown environments are some new abilities that are expected from the autonomous mobile robots. Mobile robots are no longer bound to indoor applications. Furthermore, they can even be meant to operate in unknown or changing environments. In these cases, robots can no longer be preprogrammed to pursue a fixed course of action like traditional factory robots. Instead, they must plan their actions according to the environment and react to potential unexpected situations [2].



Figure 1.2 : A man searching for a mine

One recent example project for mobile robots is the Life in the Atacama (LITA) project (figure 1.3). This project seeks to develop technology in support of robotic astrobiology for NASA while conducting useful Earth science in the Atacama Desert of northern Chile [3]. The Atacama is one of the driest places on Earth, and has long been known to support very little life. The project intends three seasons of rover field-testing and biological investigation, and culminates in a multi-week field demonstration in which scientists in the United States will direct a robotic search for life in Chile. The robots mission will be to characterize the presence and distribution of microscopic life over more than 100 kilometers of travel in an effort to better understand the limitations of life in the Atacama ecosystem. The goals of the LITA field investigation dictate traverses between distant science survey sites, opportunistic science, and extended 24-hour autonomy, including re-configuration for hibernation during the night. The mission profile is an ideal application of

Mission-Level Planning navigational autonomy that considers the long-range effects of actions in terms of route, timing and resources. In contrast to local navigation that considers spans of meters and minutes at high resolution, mission-level path planning considers hundreds of meters and hours at lower resolution. Rather than planning for locomotion exclusively, a mission-level path planner considers how certain other events, such as stationary solar charging or hibernation, affect a day's mobility activities. In bridging the gap between path planning and classical planning and scheduling, a mission-level navigation planner could provide a coarse framework, based on an integrated approach, from which to derive more detailed plans [4].

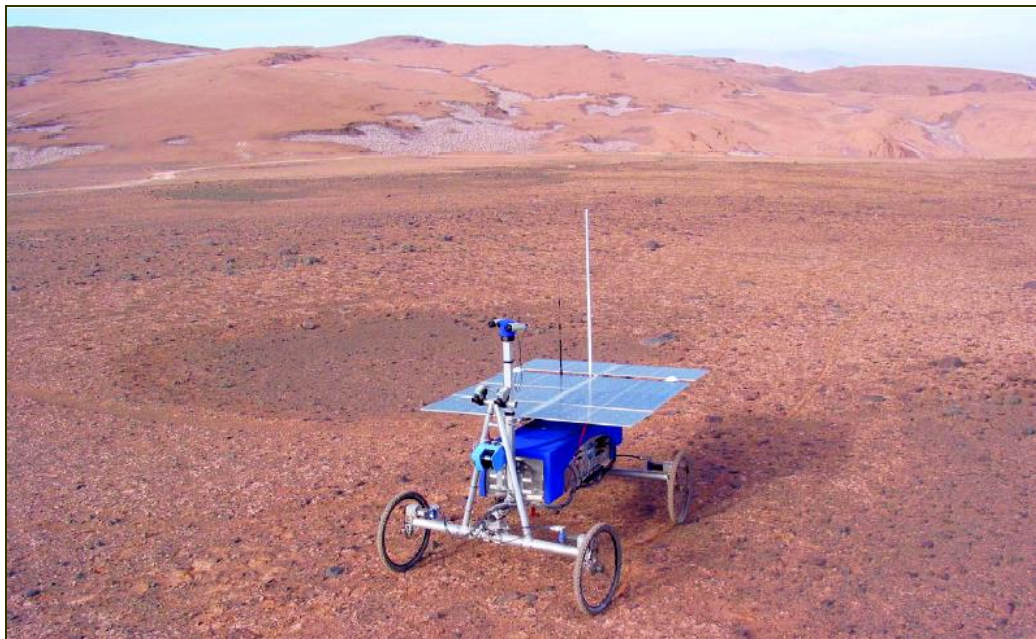


Figure 1.3: Hyperion in the Salar Grande, Atacama Desert, April 2003.

To reach a reasonable degree of autonomy, map building and navigation are very important for mobile robots. In the map building process sensors information are collected through the robot exteroceptive sensors about the environment at a given robot position and processed in order to build a local representation of the surrounding area. This representation is then integrated in the global map so far reconstructed by filtering out insufficient or conflicting information. The basic steps for map building process is as follows:

Perception: The robot sensors (in our case it is ultrasonic sensors) are activated in a proper sequence and a packet of range readings are collected.

Processing: Ultrasonic measures are processed in order to build a local representation of the surrounding scene in terms of empty and occupied space.

Fusion: The local representation is integrated in the global one by filtering out contradictory and insufficient information. Unexplored areas are regarded as dangerous in motion map, while they are considered to be safe in planning map.

Once a reasonable representation of the environment is obtained, the vehicle needs to be controlled to perform a certain task like reaching to the goal. The navigation process generates robot motions on the basis of the information provided by the map building one. It prescribes the two following phases.

Planning: A* or D* computes a path from the current robot position to the goal on the planning map. This path will be safe inside the area explored so far, and will aim directly at the goal outside.

Motion: The planned path is followed as long as it is safe on the motion map, up to the boundary of the explored area. This phase is aborted if the proximity sensors detect unexpected obstacles that obstruct the motion [5].

Navigation consists of answering three simple questions:

- i) where am I?
- ii) where am I going?
- iii) how do I get there?

The answer to the first question is known as the localization problem. It involves determining the agent's position according to what it perceives and where it was previously believed to be. This problem is typically solved by measurement, correlation and triangulation. The second and third questions involve determining a goal and planning a path that leads to the goal. They basically concern path planning and collision avoidance [6].

In the navigation phase, an A* or D* based planner generates a path from the current robot position to the goal. Such a path is safe inside the explored area and provides a direction for further exploration. The robot follows the path up to the goal, terminating its motion if unexpected obstacles are encountered. Map building and navigation are alternately performed during the task execution (see Figure. 1.4). It is easy to understand that, if the environment is static, then mobile robot can find the

goal if it exists or returns fail otherwise. In fact, if a solution exists, the explored area is increased until it contains the goal. At this point, the completeness guarantees that a path inside this area is found. Conversely, if the problem does not admit a solution, the robot will explore all the connected region that can be reached from the start position. When the map of this area is completed, the planner will return a failure.

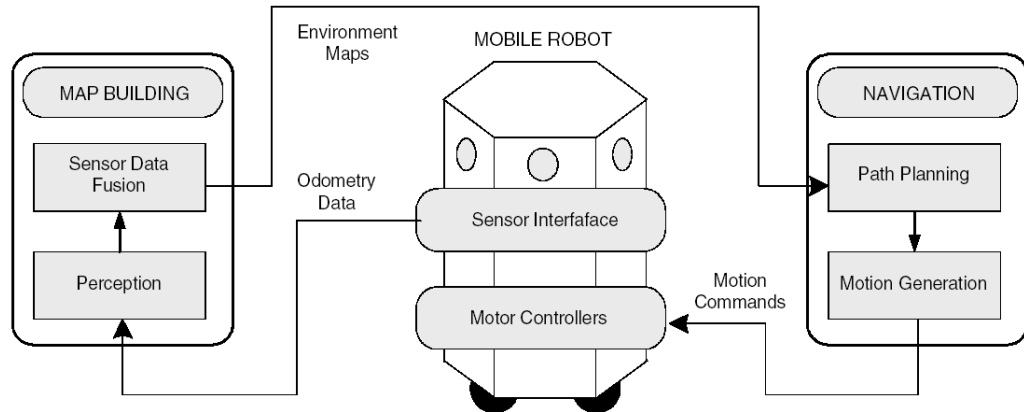


Figure 1.4: Functional diagram of the map building and navigation processes

The problems that we need to deal for mobile robots are listed below:

- i) Map building
- ii) Robot localization
- iii) Path planning.
- iv) Collision avoidance

As a solution for the map building problem grid based mapping, quadtree mapping of framed-quadtree mapping can be used. Grid based maps [7, 8, 9] are generated from the occupancy state of an object in a given cell. It is rather easy to construct and maintain them with reasonable computing power in small workspace. However, in large environments the storage space and the processing of the maps become matters of concern. The object boundaries can be approximated with straight lines or curves from additional processing of the sensor measurements. A method of building a map of the environment for mobile robot equipped with a radial laser scanner was proposed by Gonzalez [10, 11]. The use of fuzzy set to include any uncertainty in the actual place of the object boundaries has also been suggested by some of the

researchers [12,13]. The above existing approaches are used high accurate sensor like laser scanner. And in some approaches ultrasonic sensors are used. It is assumed that the world can be adequately modeled in two dimensions. Just as humans use two-dimensional floor plans, the robot uses a map, which projects all features into two dimensions. The complexity of the map is thereby greatly reduced. The simplification is often reasonable in practice, especially in man-made environments [14].

Methods that use uniform grid representations must allocate large amounts of memory for regions that may never be traversed, or contain any obstacles. Kambhampati and Davis [15] introduced hierarchical path-searching using quadtrees, but the method produces suboptimal paths and suffers from the fundamental problem of misrepresentation of terrain data in quadtree-nodes. Zelinsky [16] proposed strategies to smooth paths found using quadtrees, which include the use of a quadtree-based visibility graph, but the resulting paths are not optimal to the resolution of the smallest cell. Recently, a new data structure called a framed quadtree has been suggested by Yahja et. al. [17] in 1998 as means to overcome some of the issues related to the use of quadtrees. Framed-quadtrees address both problems, especially in sparse environments; it's paths are always shorter, and in all but the most cluttered environments it executes faster and uses less memory than when regular-grids are used. In general, the sparser or the more unknown the world, the greater the advantage of using framed-quadtrees [6].

The localization problem is not easy to solve. Basically, most systems rely, at least partly, on odometry. However, robot slippage provokes small positioning errors that accumulate unrestrainedly. Consequently, after a while the robot may not know its real position. The problem is even worse if no odometric information is available. In this case, the problem is known as global localization, while localization based on odometric information is known as tracking. Most tracking approaches rely on Kalman filtering to integrate sensor information over time. Global techniques aim instead at locating significant features in the environment in order to determine the robot position with respect to those landmarks. Further information on localization can be found in [18]. This work focuses on techniques to plan how to get goal so localization information is assumed as already valid [6].

Most autonomous outdoor path planning solution tested on actual robots have centered on local path planning tasks such as avoiding obstacles or following roads. Global path planning has been limited to simple wandering, path tracking, straight-line goal seeking behaviors, executing a sequence of scripted local behaviors or following a specific type of terrain feature. These capabilities are insufficient for unstructured and unknown environments, where replanning may be needed to account for new information discovered in every sensor image. Mobile robots operating in vast outdoor unstructured environments often only have incomplete maps so they can be equipped with one or more sensors to measure the location of obstacles, locate itself within the environment, and check for hazards. With each sensor information addition to the map, the global system uses an incremental path planning algorithm to accommodate new information, to optimally replan the global path and recommend steering commands to reach the goal. Existing approaches plan an initial path based on initial map information and then modify the plan locally or replan the entire path as the robot discovers obstacles with its sensors, sacrificing optimality or computational efficiency respectively. It is possible to replan a new global trajectory for each new piece of sensor data, but in cluttered environments the sensors can detect new information almost continuously, thus precluding real-time operation. Furthermore, sensor noise and robot position error can lead to the detection of incorrect location of obstacles, flooding the global navigator with more, and sometimes erroneous data. The optimal algorithms search a state space using the distance transform to find the lowest cost path from the robot's start state to the goal state. Cost can be defined to be distance travelled, energy expended, time exposed to danger, etc [19].

One of the approaches is to generate a “global” path that uses the unknown information then attempt to “locally” circumvent obstacles on the route detected by the sensors [20]. If the route is completely obstructed, a new global path is planned. Lumelsky [21] initially assumes the environment to be devoid of obstacles and moves the robot directly toward the goal. If an obstacle obstructs the path, the robot moves around the perimeter until the point on the obstacle nearest the goal is found. The robot then proceeds to move directly toward the goal again. Pirzadeh's [22] strategy is the robot wanders till it reaches to the target. The robot moves to the adjacent location with the lowest cost and increments the cost of a location at

everytime it visits the same space. Korf [23] uses initial map information to estimate the cost to the goal for each state and efficiently updates it with backtracking costs as the robot moves through the environment. While these approaches are complete, they are suboptimal in the sense that they do not generate the lowest cost path given the sensor information as it is acquired and assuming all known, a priori information is correct. It is possible to generate optimal behavior by computing an optimal path from the known map information, moving the robot along the path until either it reaches the goal or its sensors detect a discrepancy between the map and the environment, updating the map, and then replanning a new optimal path from the robot's current location to the goal. A* [24] algorithm can overcome these problems but it can be grossly inefficient, particularly in expansive environments where the goal is far away and little map information exists. Boulton [25] maintains an optimal cost map from the goal to all states in the environment, assuming the environment is bounded (finite). When discrepancies are discovered between the map and the environment, only the affected portion of the cost map is updated. The map representation is limited to polygonal obstacles and free space. Trovato [26] and Ramalingam and Reps [27] extend this approach to handle graphs with arc costs ranging over a continuum. The limitation of these algorithms is that the entire affected portion of the map must be repaired before the robot can resume moving and subsequently make additional corrections. Thus, the algorithms are inefficient when the robot is near the goal and the affected portions of the map have long "shadows". Stentz overcomes these limitations with D*, an incremental algorithm which maintains a partial, optimal cost map limited to those locations likely to be of use to the robot. Likewise, repair of the cost map is generally partial and re-entrant, thus reducing computational costs and enabling real-time performance. D*, algorithm is capable of planning paths in unknown, partially known, and changing environments in an efficient, optimal, and complete manner [28].

Robot path must be safe inside the area so far explored, and at the same time should provide directions for further exploration aimed at reaching the goal. This is realized by defining cost functions that characterize the risk of collision along a path, and by choosing a proper instance of the class of graph search algorithms in order to obtain a minimum-cost path. In fact, while it is possible to devise general control architectures that behave robustly in various situations, in this way one might be

forced to give up interesting formal properties such as completeness, that can instead be guaranteed by algorithmic approaches. Further advantages of these are the possibility of analyzing complexity as well as the efficiency of the obtained paths.

In this work path planning algorithms (A^* , D^*) and map building techniques (regular grids, quadtrees and framed quadtrees) are compared by using matlab program. This work is organized as follows. In Section 2 modelling of outdoor environment is described with map building techniques and in Section 3 path planning methods are given. In Section 4 Sensor model is described for simulation studies and in Section 5 simulation results are given. Finally, this studie is concluded in Section 6.

2. MODELLING OF OUTDOOR ENVIRONMENT

In order to evaluate different strategies for navigating in an uncertain environment, a specific type of robot environment for simulation is chosen and called as realmap (figure 2.1). Without a loss of generality, assumed that the robot's environment is a two-dimensional, eight-connected grid. Each grid cell contains a positive cost value representing the per unit cost of moving across the cell. Moving horizontally or vertically cost 1 unit and moving along the diagonal costs 1 times square root 2. In framed quadtree costs are calculated according to the euclidean rules. $32*32$, $64*64$, $128*128$ and $256*256$ cells are used for simulations and the obstacle distribution is shown below. The robots starting point is defined as (1,1) and the end point is defined as (rows,cols).

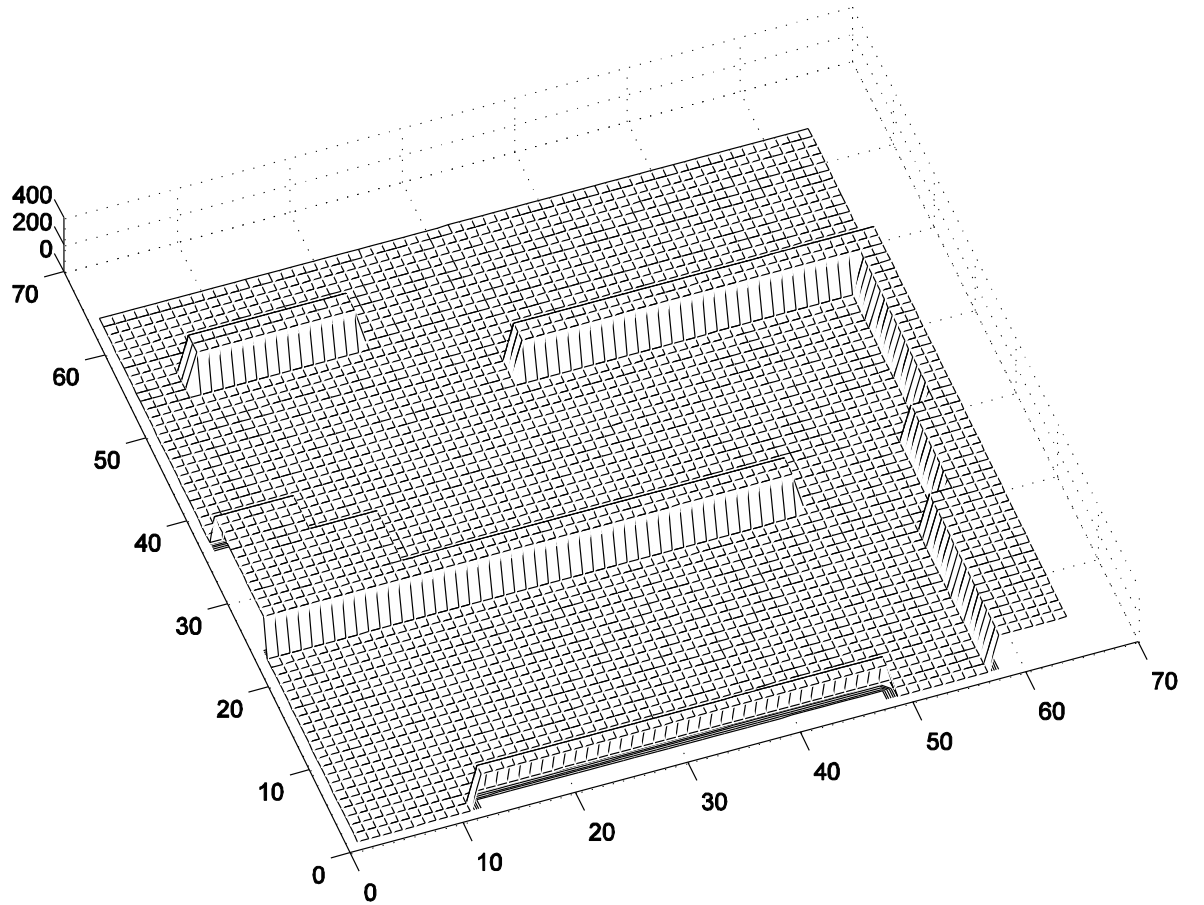


Figure 2.1: Realmap generated by Matlab for simulation ($64*64$)

Each grid cell (x, y) in the map yields the occupancy probability of the respective region of the environment. These cells are modified according to the readings of Polaroid sonar sensors, which have an arc of uncertainty of approximately 25 degree. To build the metric map, sonar readings are integrated into a local occupancy grid, which is the result of the weighted addition of all the sonar sensor models. According to the sensor measurements the local grid has zero and 255 values, which correspond to empty and occupied regions, respectively. Sonar interpretations must be integrated over time to update the map coherently.

Finally, it is important to note that the precision of the resulting metric map depends on the correct alignment of the robot with its map. Hence, slippage and drifting must be detected and corrected. This information is extracted from the localization layer, which uses well-known techniques like correlation of the local map and the respective section of the global map.

Unless the robot's environment is completely known, there exists one or more cells for which the obstacle must be estimated. There are innumerable strategies for estimating the obstacles situation being or not, three of them are described below:

Optimistic strategy: each cell for which the obstacles situation is unknown is assumed to be of lowest cost. For the environment model above, this corresponds to level, easy-to-traverse terrain.

Pessimistic strategy: each cell for which the obstacle is unknown is assumed to be of highest cost but not 255 which is obstacle. For the environment model above, this corresponds to the steepest terrain that is still traversable.

Average value strategy: each cell for which the cost value is unknown is assumed to be equal to the average of known cells in the vicinity.

Best case and worst case approaches due to optimistic and pessimistic scenarios can be obtained one after another. Cost distribution's first moment is assumed as a good estimation of the value of an individual member in the average value strategy. There is no obstacle in an unknown cell is an assumption for the above strategies. The path planning algorithm could not assume the path to the goal correctly in case putting an extra obstacle in a wrong place in the map. The only passageway to the goal is assumed to be closed by the obstacle is the case for incorrect path planning.

According to optimistic strategy, the unknown cells are assumed as empty for this work [29].

Space discretization is a facility for search method implementation and provides a flexible representation for obstacles and cost maps as well as control allowance over the complexity of path planning. Tessellation of space into equally sized cells having connection to their eight neighbours is one method of cell decomposition. Higher memory requirements and suboptimal resulting paths are the two disadvantages of this method. Although there are regions that may never be traversed or that may not contain any obstacles, uniform grid representation methods use so much memory for these regions. Efficiency in map representation which is the first problem can be solved by using quad trees, but at a cost of optimality. More optimal paths will be obtained by using framed quad trees as a solution of latter problem. Below example is given for comparing the representations [17].

2.1. Regular cells

Regular grids are easy to implement and especially for indoor applications time for map building and path planning algorithms is very short. One of the drawbacks of regular grids is that they represent space inefficiently. Natural terrain is usually sparsely populated and is often not completely known in advance. Many equally sized cells are needed to encode empty areas, making search expensive since a very large number of cells must be searched. Moreover, regular-grids allow only eight angles of direction between cells, resulting in abrupt changes in path direction and an inability, in some cases, to generate straight paths through empty areas (Figure 2.2). It is possible to smooth such paths, but there is no guarantee that the smoothed path will converge to the optimal path.

Map building for regular grids is very easy, simply divide the area into equal parts and path planning algorithm the cells are reachable by directly through the cost and direction matrices. During the movement of the mobile robot the information comes from sensors are used for updating these direction and cost matrices update.

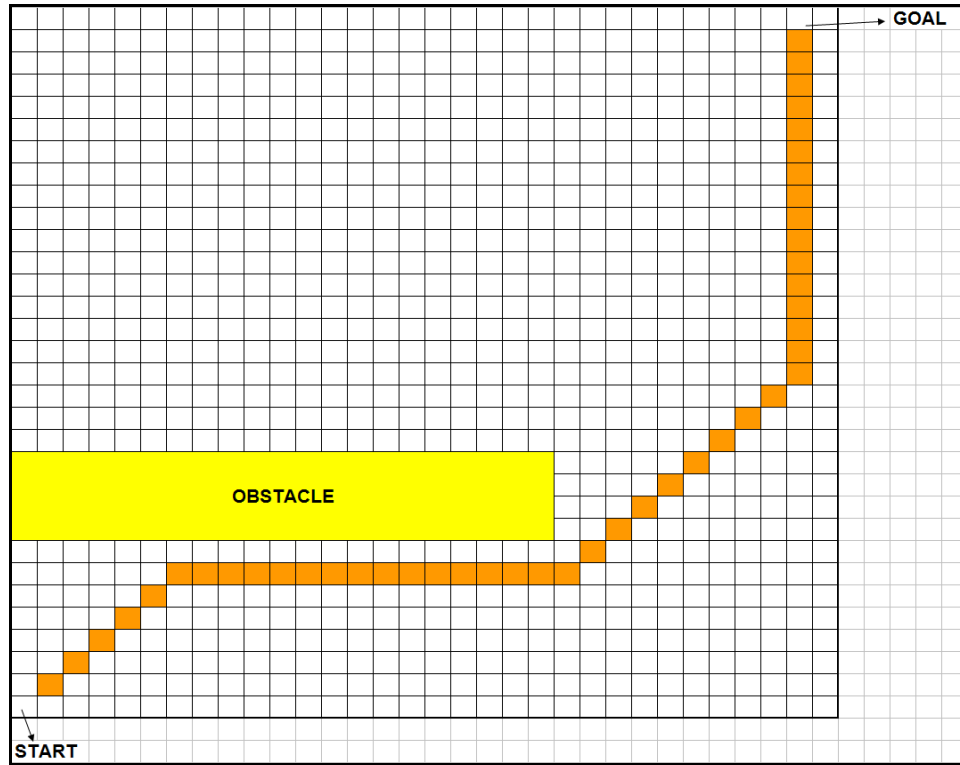


Figure. 2.2: Outdoor modelling using regular cells.

The cells in the real map are divided into 9 different states. For example for the cells which are closed to left side (2) it is not possible to move left so, for these cells only the right, top and bottom sides are searched. In simulation program it is easy to address cells by direction and cost matrices for regular grid maps. As it is discussed in detail in path planning part, according to the position of the cell in the map simulation searches for the next step. Searching starts with the aim cell.

Table 2.1: Different states for regular map calculation in Matlab program

1	4	4	4	7
2	5	5	5	8
2	5	5	5	8
2	5	5	5	8
3	6	6	6	9

The nearest cells which has no obstacle are listed in open list and direction and cost values are calculated for openlist. The cells in the openlist are taken as a aimcell one by one and the nearest cells which has no obstacle and which has no cost values are listed in new openlist and direction and cost values are calculated for this new openlist. This searching algorithm continues till current state of the robot is added in to the openlist. By using the direction values of the cells robot starts to move until it

detects any obstacle by its sensors. When it detects an obstacle the robotmap updated and the searching algorithm `direction_regular` starts again.

2.2. Quadtree cells

One way to reduce memory requirements is to use a quadtree instead of a regular-grid. Quadtrees [30] are created by recursively subdividing each map square with non-uniform attributes into four equal-sized sub-squares. The process is repeated until a square doesn't contain any obstacle or the highest resolution of the map is reached. In the latter case, a highest-resolution cell is marked as an obstacle cell if it contains any obstacle. Figure 2.3 shows the quadtree subdivision of the map area and the corresponding quadtree data structure. The leaves (i.e., quadtree-nodes without children) are called terminal quadtree-nodes. A quadtree with a single top-level node is called a single-root quadtree. A collection of connected single-root quadtrees forms a multiple-root quadtree or a quadforest.

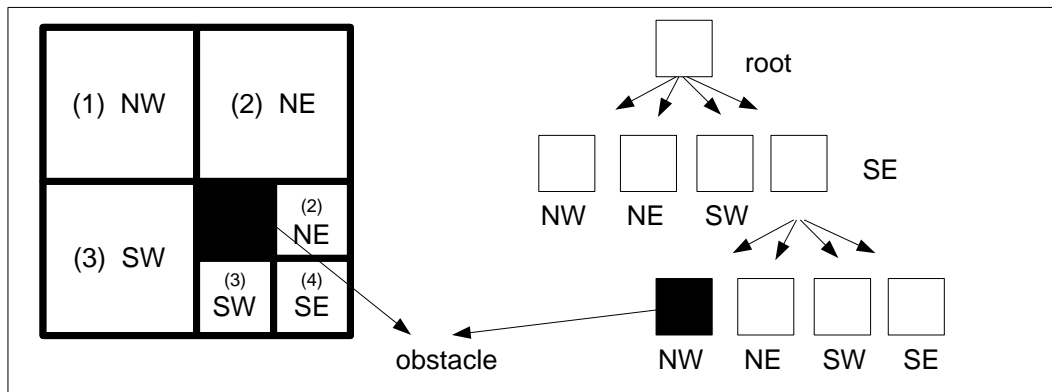


Figure 2.3: Quadtree tessellation of a region due to a single obstacle and the corresponding quadtree data structure.

The quadrants are labelled as NE (Northeast), NW(Northwest), SW (SouthWest), and SE (Southeast). Quadtrees allow efficient partitioning of the environment since single cells can be used to encode large empty regions. However, paths generated using quadtrees are suboptimal because they pass through the center of each quadtree-node. There are other strategies to construct and smooth paths for quadtrees [16] so that the paths do not necessarily pass through the center of each quadtree-node, but there is no guarantee that they are optimal to the highest cell-resolution. Figure. 2.4 shows an example path generated using a quadtree [17].

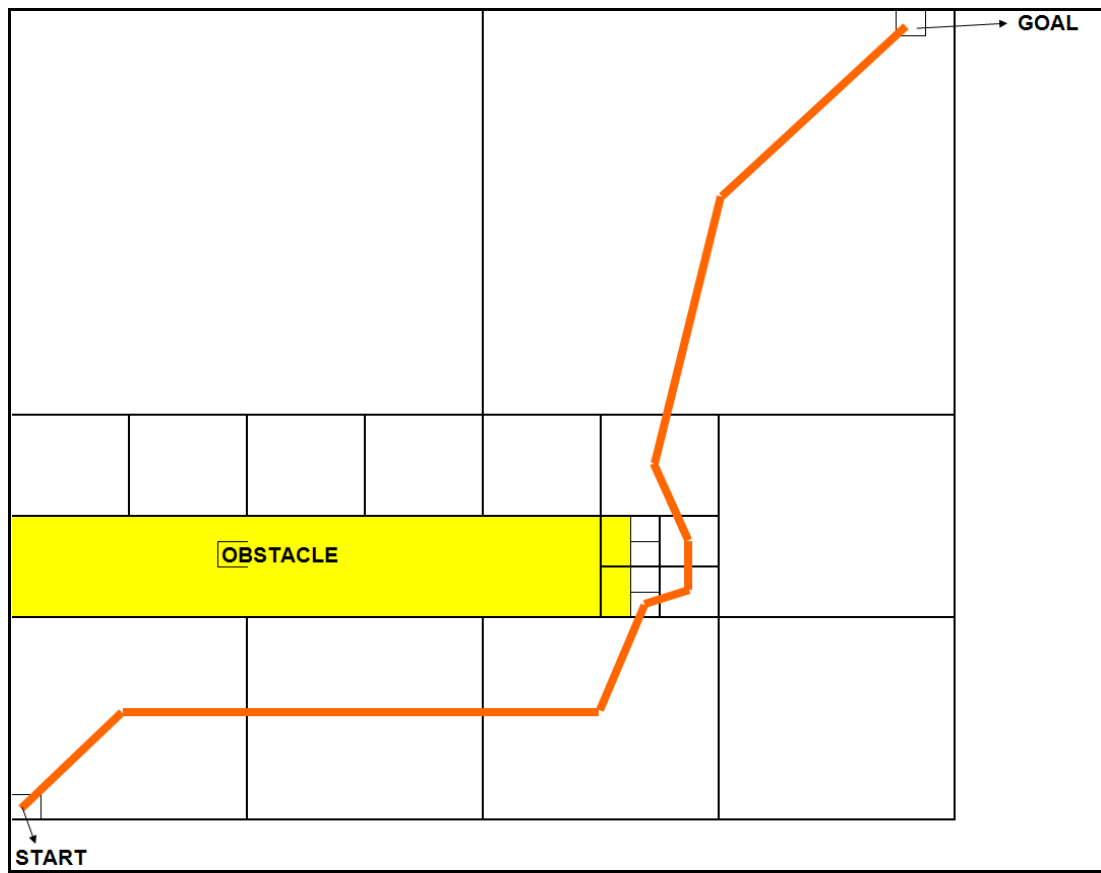


Figure 2.4: An example of a path generated using quadtrees

2.3. Framed-Quadtree cells

To obtain smooth paths a modified data structure in which cells of the highest resolution are added around the perimeter of each quadtree region [31]. An example path generated using framed-quadtree is shown in Figure 2.5. The small rectangles around each quadtree region are the “border” cells of each framed-quadtree node. As paths can move through any border-cell via straight lines between border-cells and there are many border-cells inside large framed-quadtree nodes, this representation permits many more discrete angles of direction between cells, instead of just eight angles, as in the case with regular grids. Most importantly, the paths generated more closely approximate optimal paths. The drawback of framed-quadtrees is that they can require more memory than regular-grids in highly cluttered environments because of the header information involved in the book-keeping [17].

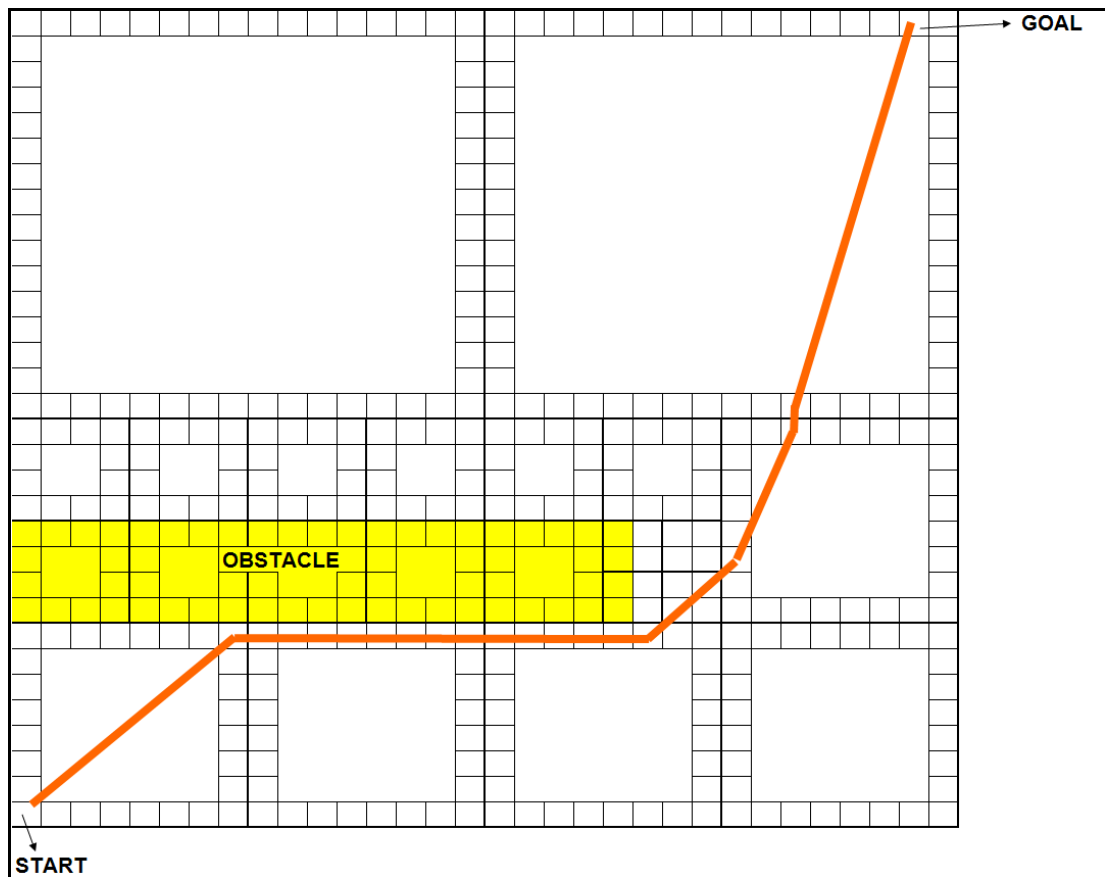


Figure 2.5: An example of a path generated using framed-quadtree.

Map building of framed-quadtree is based on header file instead of simple matrices used for regular grids. Instead of using all the smallest cells in an area to save memory framed-quadtree uses only the cells which are on the way. So the cells which are needed are defined in header file with the information of parent, quadtree node, division level, childtype of quadtree node and cell position in the quadtree node.

Data structure used for framed quadtrees is given below with Figure 2.6, every cell keeps this information as a header file.

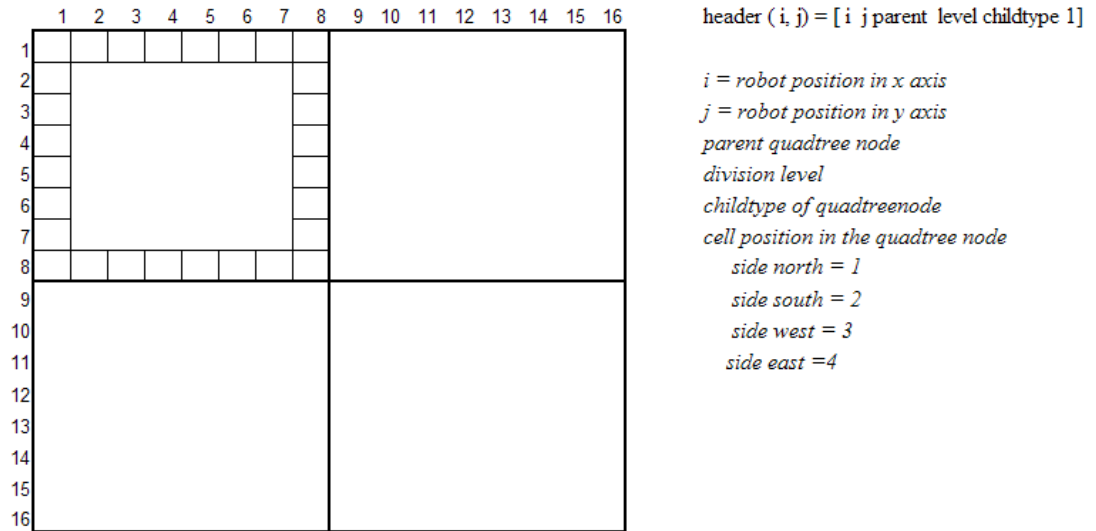


Figure 2.6: Data structure for framed quad-trees

Divide function splits the map for once into four equal squares and updates header file with the new border cells.

Main program looks for obstacle positions if they are listed in the header file or not, if not then main program divides the parent cell that obstacle belongs to and with this information main program calls divide function.

Main program calls divide function with division level and parent quadtree node information till all the obstacles are defined in header file.

The splitting of a quadtree-node also causes the modification of framed-quadtree structures by re-executing the framed-quadtree build-up procedures incrementally and locally, this can be done by calling divide function (Figure 2.7). According to division level and parent quadtree node division is done.

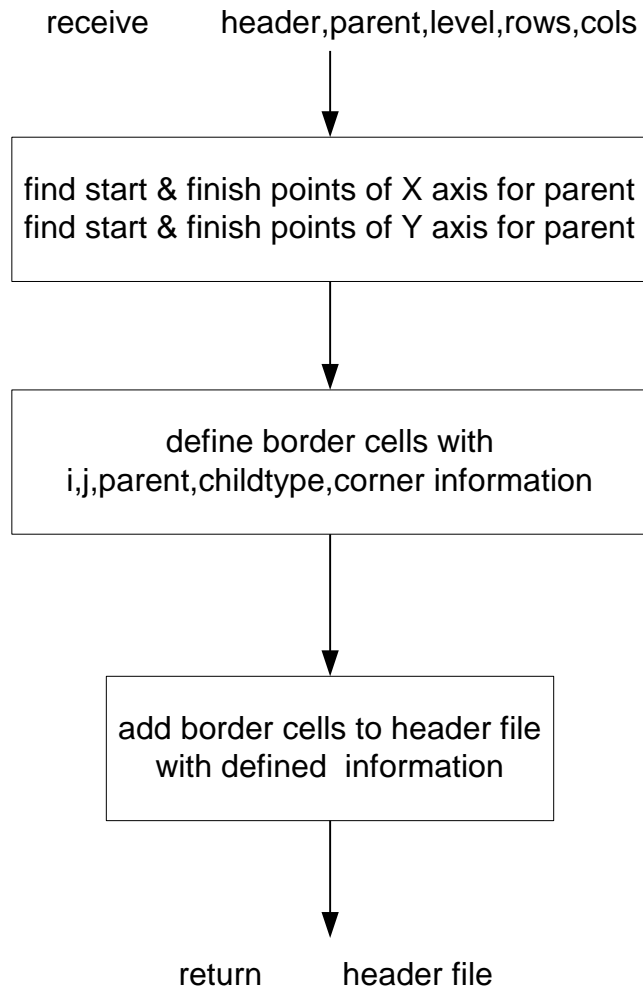


Figure. 2.7: Divide function for framed-quadtree structure

After creating the map the path planning algorithm calculates direction and cost functions using header file.

As discussed above, if a new obstacle is detected inside an empty quadtree-node, we split that node into 4 subnodes as illustrated in Figure 2.8. New border cells should be added to the header file and the current cells should be deleted to avoid having the same cell with different information

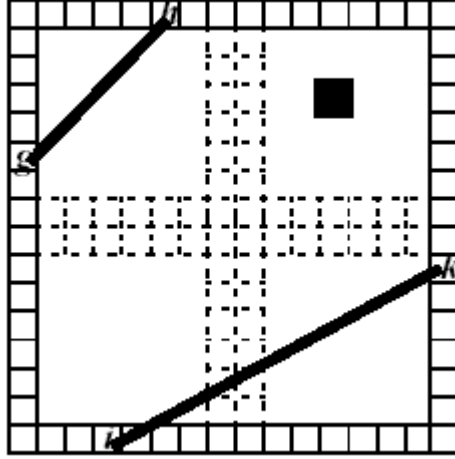


Figure 2.8: As the splitting of a framed-quadtree node occurs (the dotted rectangles show the first step of splitting), some links like edge gh are not affected, while others like edge ik are affected.

In this way, the framed-quadtree operation mimics the local and incremental nature of D^* , allowing efficient computation. The proposed approach is locally adaptable, that is, it changes the framed-quadtree structures and applies the D^* algorithm locally in response to some local environment change, while maintaining global path optimality.

3. PATH PLANNING

Path planning for a mobile robot is typically stated as getting from one place to another. Mobile robot efficiently reach its goal while successfully navigating around the structured world. Avoiding colliding with an obstacle such as a rock is one of the challenges for the structured world that is often found indoors. Additionally special challenges occur in outdoor environments like avoiding falling into a pit or ravine and avoiding travel on terrain that would cause it to tip over. Also, the range of obstacles that can interfere with the robot's passage is large; the robot must still avoid a rock as well as go around a mountain. Large areas are mapped at high resolution and collects information while mobile robot is moving and updates its map. Because of this the solution for path planning must be incremental. Another challenge is dealing with a large amount of information and a complex model of the vehicle. Taken as a single problem, so much information must be processed to determine the next action that it is not possible for the robot to perform at any reasonable rate. Because of this information researchers generally deal with this issue by using a layered approach to navigation like seperating it into two levels: local and global. Local planning is for avoiding obstacles, reacting to sensory data as quickly as possible while driving towards a subgoal [32]. Global planning, operating at a coarser resolution of information is used to decide how best select the subgoals such that the goal can be reached. Approaches to path planning for mobile robots can be broadly classified into two categories; those that use exact representations of the world, and those that use a discretized representation. The main advantage of discretization is that the computational complexity of path planning can be controlled by adjusting the cell size. In contrast, the computational complexity of exact methods is a function of the number of obstacles and/or the number of obstacle facets, which we cannot normally control [17, 33].

The path is optimal in path length if the sum of the transition costs, also called arc costs, is minimal across all possible sequences through the graph. Finding the lowest-cost path through a graph is central to path planning for a mobile robot. As the robot

acquires sensor data, it can update its map and replan the optimal path from its current state to the goal. It is important that this replanning be fast, since during this time the robot must either stop or continue to move along a suboptimal path.

Unstructured outdoor environments are often sparse and also the resolution they have been mapped is coarse. If complete and accurate maps were available, it would be sufficient to use a standard search method such as A* [34] to produce a path. Imperfections in control, inertial sensing, and perception often introduce erroneous and changing information. A mobile robot must gather new information about the environment and efficiently replan new paths based on the information received from sensors. This approach, known as Best Information Planning, produces a path based on all available information and replans from the current position to the goal when new information becomes available. Best Information Planning is able to make use of prior information to reduce the traversal cost. It is possible to use A* to replan a new path, but this approach is computationally expensive because it must replan the entire path to the goal every time new information is added. Another approach is to use D*, an algorithm suited for Best Information Planning, that allows replanning to occur incrementally and optimally in real-time. In other words, in contrast to A*, D* does not require complete replanning of path every time new information comes in. Incremental replanning makes it possible to greatly reduce computational cost, as it only updates the path locally, when possible, to obtain the globally optimal path. D* produces the same results as planning with A* for each new piece of information, but is much faster. The reason for this is that D* adjusts optimal path costs by increasing and lowering the costs only locally and incrementally as needed. D* is the dynamic version of A*, which maintains optimality, unlike other distance/path transform planners. Like A*, D* operates on a cost graph. D* maintains a list of states (or graph nodes) queued for cost expansion (that is, cost recalculation and propagation), initially with the goal state put on the list with a cost of zero. This list is called the open list, as it contains states “open” for expansion. The state with the minimum path cost on the open list is repeatedly expanded, propagating path cost calculations to its neighbors. As the robot moves, it may detect new obstacles or the absence of expected obstacles. When it detects an obstacle, the arc of the path passing through this obstacle is marked with 255 value and the adjoining states are put on the open list for cost correction. Encountering unexpected obstacles will set off a “raise”

wave, a wave of increasing cost, through neighboring states. When this wave meets with states that are able to lower path costs, these “lower” states are put on the open list to recalculate new optimal paths. When it detects the absence of an expected obstacle, the arc of the path passing through this “missing” obstacle is marked with 0, denoting an empty space, and the adjoining state is put on the open list as a lower state, setting off a “lower” wave, a wave of decreasing cost. D* is able to determine how far the raise and lower waves need to propagate while providing the optimal path for robot traverse continuously [17].

3.1. The A* Algorithm

Consider the following approach for using map information during the robot's traverse. Let S be the robot's start state, G the goal state, X the current state, and M the robot's current map. Steps in the A* algorithm:

- 1.Store all known, approximated, estimated, and believed information about the robot's environment in M. Let X equal S.
- 2.Plan the optimal path from X to G using all information in M. Terminate with failure if no path can be found.
- 3.Follow the path from Step 2 until either G is reached (terminating with success) or the robot's sensor discovers a discrepancy between M and the environment.
- 4.Update M to include the sensor information and go to Step 2.

In short, this approach plans the optimal path to the goal using all known information and replans from the current state whenever new or conflicting information is discovered. This replanning approach produces an optimal traverse defined as follows: A traverse is the sequence of states visited by the robot enroute from S to G. A traverse is optimal if, for every state X in the sequence, the successor state to X is part of an optimal path from X to G given all aggregate map information known to the robot at state X [29].

Below A* algorithm studied with regular grids and framed quadtree grids.

A* algorithm with regular grids

Consider how A* solves the following robot path planning problem with regular grids. Figure 3.1 shows an eight-connected graph representing a Cartesian space of robot locations. The states in the figure 3.1, depicted by arrows, are robot locations. The lower regions in the figure 3.2 are locations known to be in free space and the higher regions are known obstacle locations. Without a loss of generality, the robot is assumed to be point-size and occupies only one location at a time. A* can be used to compute optimal path costs from the goal, G , to all states in the space given the initial set of arc costs. The arrows indicate the optimal state transitions; therefore, the optimal path for any state can be recovered by the arrows to the goal as shown in the figure 3.1.

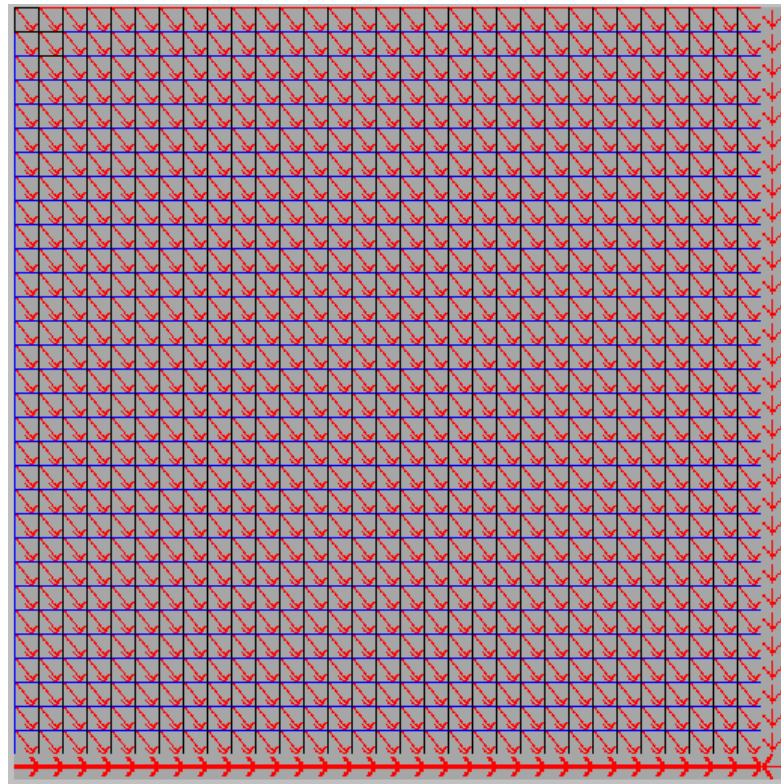


Figure 3.1: Example direction matrix for A* algorithm with regular grids, starting position

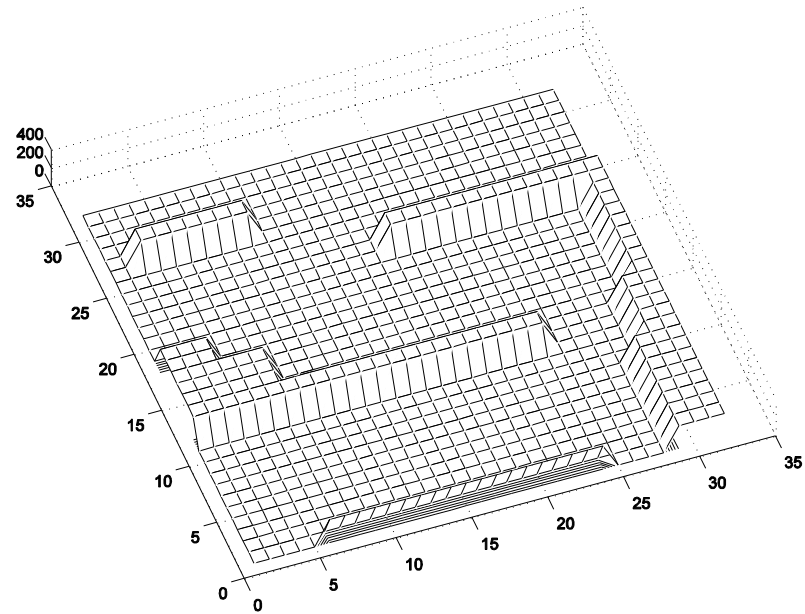


Figure 3.2: Example real map for A* algorithm with regular grids, starting position

Figure 3.3 shows the robotmap that mobile robot has at the starting position.

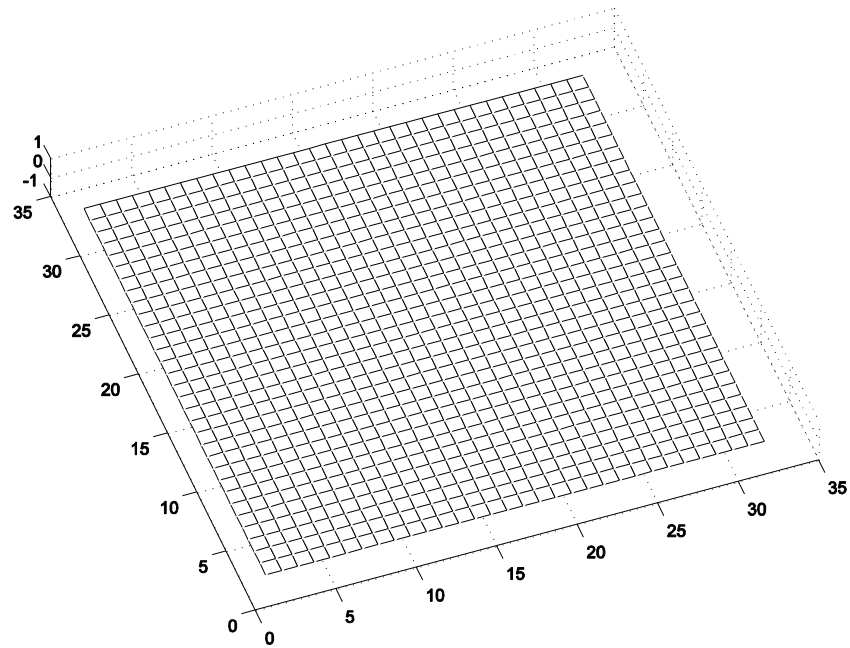


Figure 3.3:. Example real map for A* algorithm with regular grids, starting position

The robot starts at (1,1) position and begins following the optimal path to the goal. At location A, the robot's sensor discovers the obstacles shown in figure 3.4 , figure 3.5 and figure 3.6.

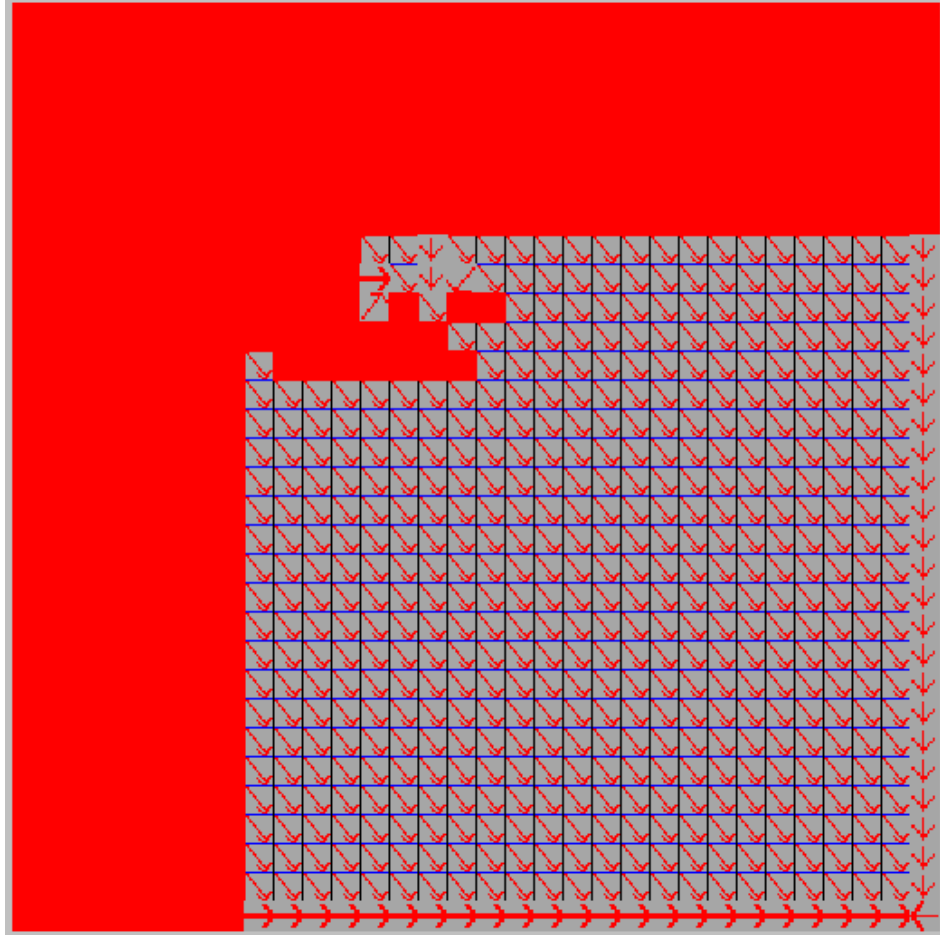


Figure 3.4: Example direction matrix for A* algorithm with regular grids, location A.

Because of the change in the robotmap A* deletes the current direction and cost matrices and starts to calculate both matrices starting with goal cell to the current position. The resulting direction vectors shown in figure 3.4.

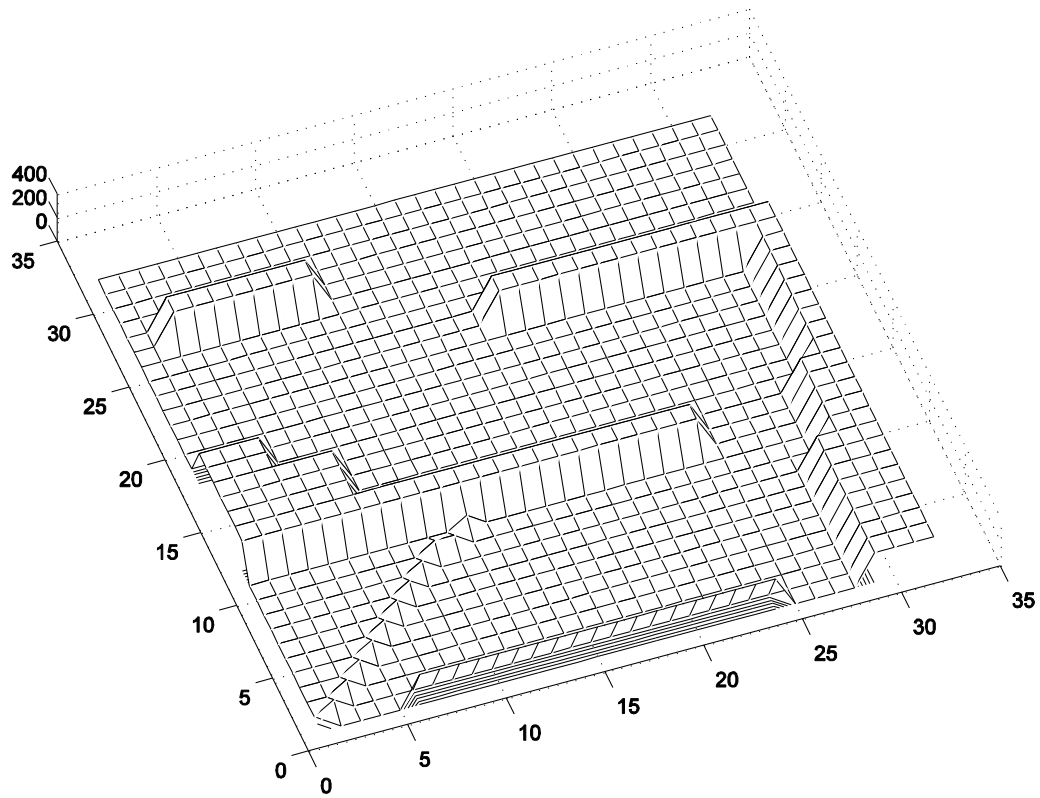


Figure 3.5: Example real map for A* algorithm with regular grids, location A.

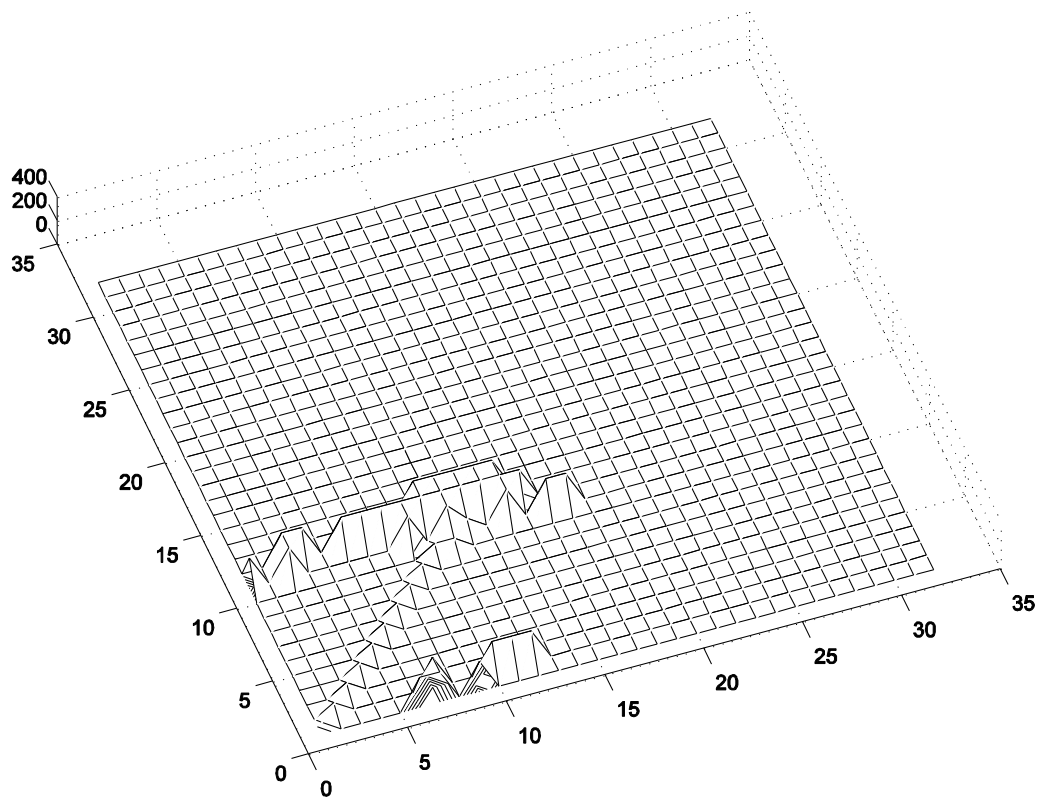


Figure 3.6: Example mobile robot map for A* algorithm with regular grids, location A.

Mobile robot continues to its movement, and in position B again find outs the other obstacles, again A* path planning algorithm updates direction and cost matrices. Direction matrices shown in figure 3.7

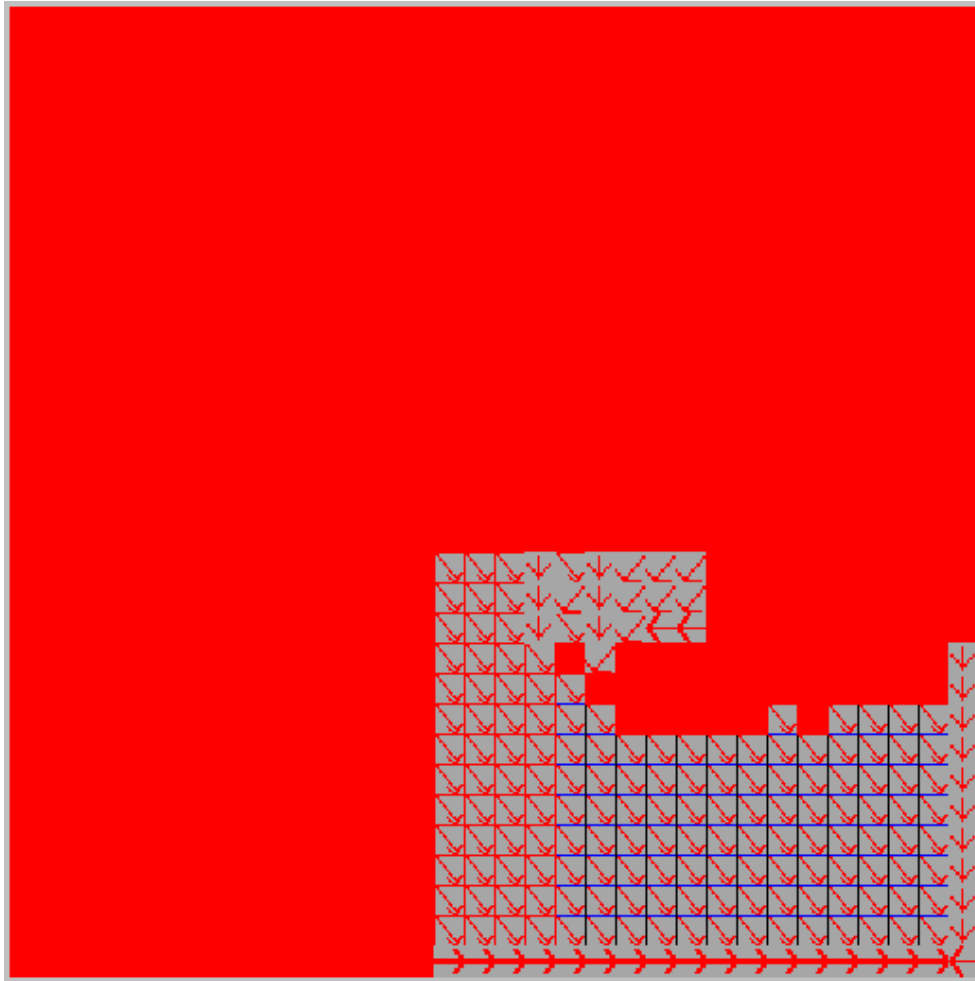


Figure 3.7:. Example direction matrix for A* algorithm with regular grids, location B.

Figure 3.8 shows the realmap that mobile robot goes, and also the obstacle positions. Figure 3.9 shows the robotmap that mobile robot has at location B.

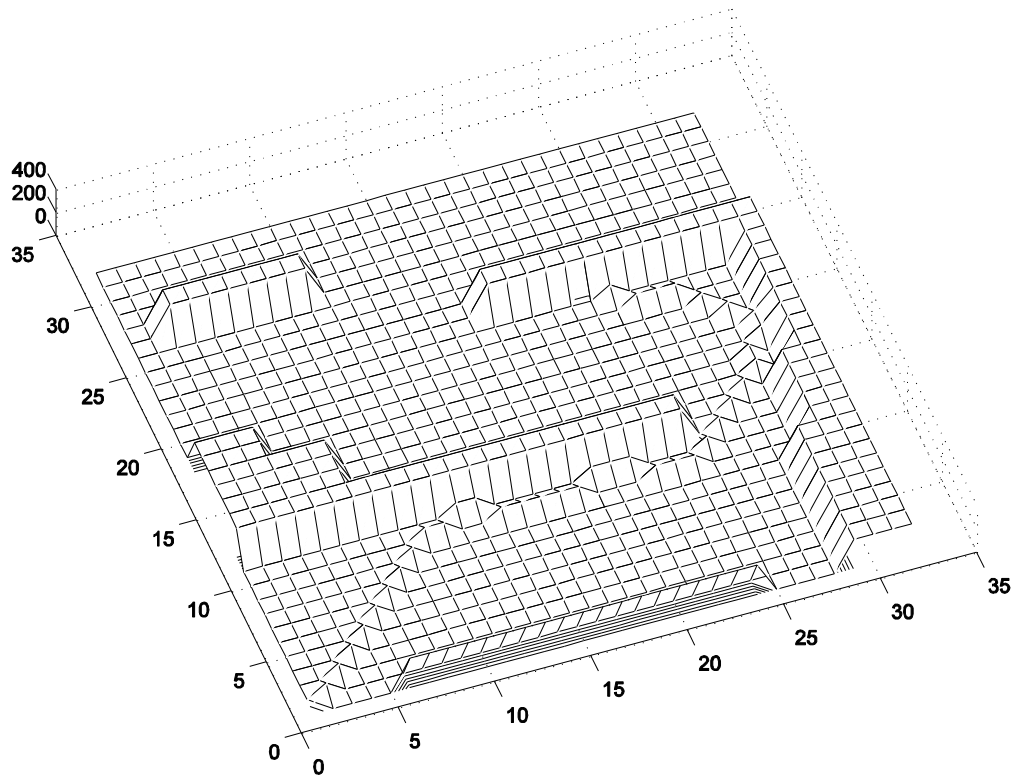


Figure 3.8: Example real map for A* algorithm with regular grids, location B.

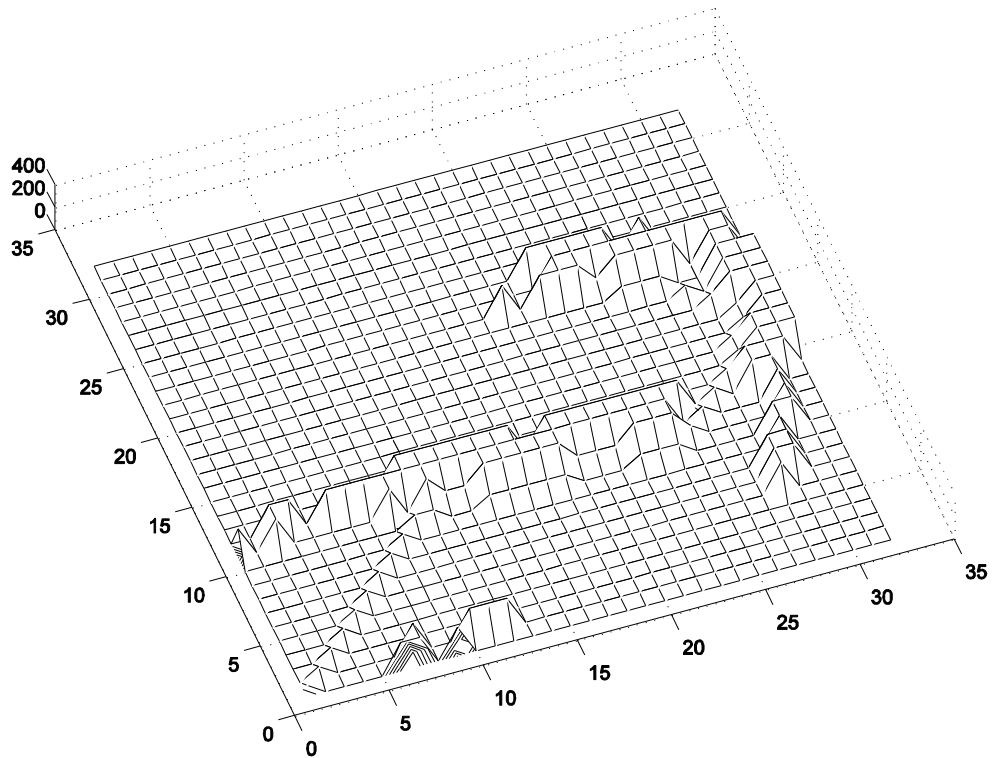


Figure 3.9: Example mobile robot map for A* algorithm with regular grids, location B

The main program flow chart for A* path planning algorithm with regular grids is given as it is shown in figure 3.10.

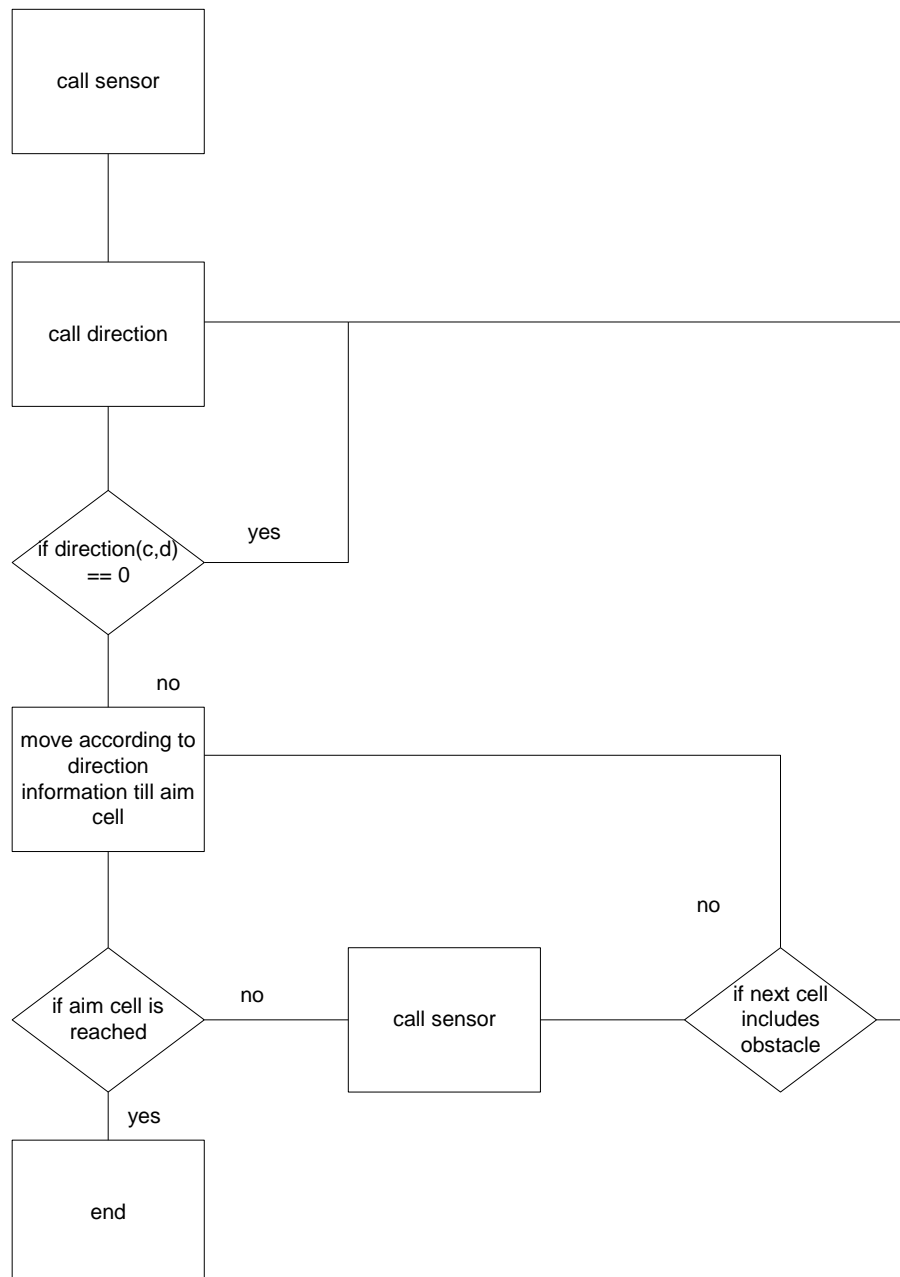


Figure 3.10: Flow chart of matlab program main A* algorithm

Flow chart of matlab program direction_regular which defines direction, cost, openlist values for regular cells is given below with figure 3.11.

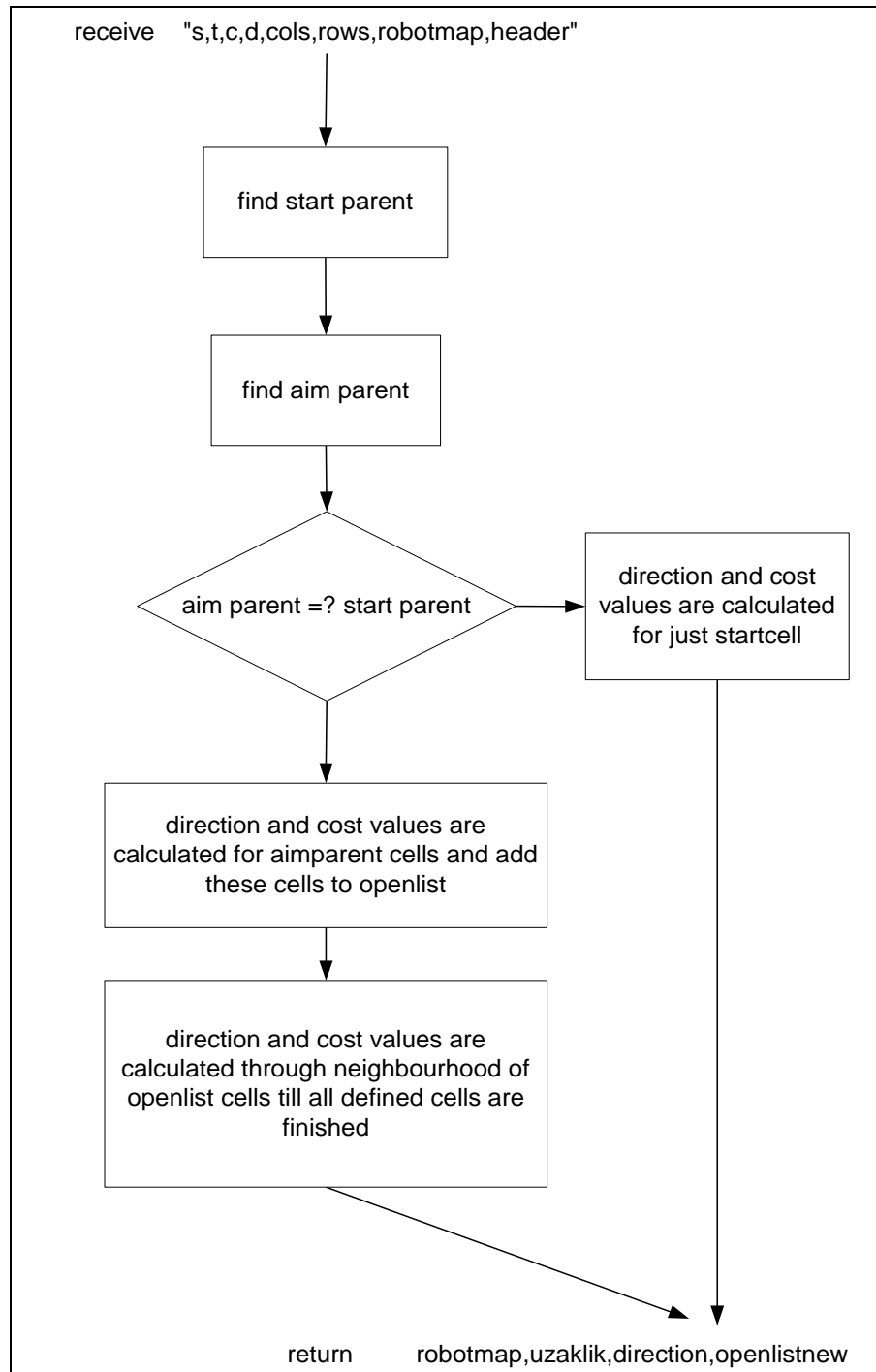


Figure 3.11: Flow chart of matlab program direction_regular

A* with framed-quadtrees

Consider how A* solves the same robot path planning problem with framed-quadtree mapping. Figure 3.12 shows the initial situation that mobile robot knows nothing about the environment, so the map has only one terminal quadtree node.

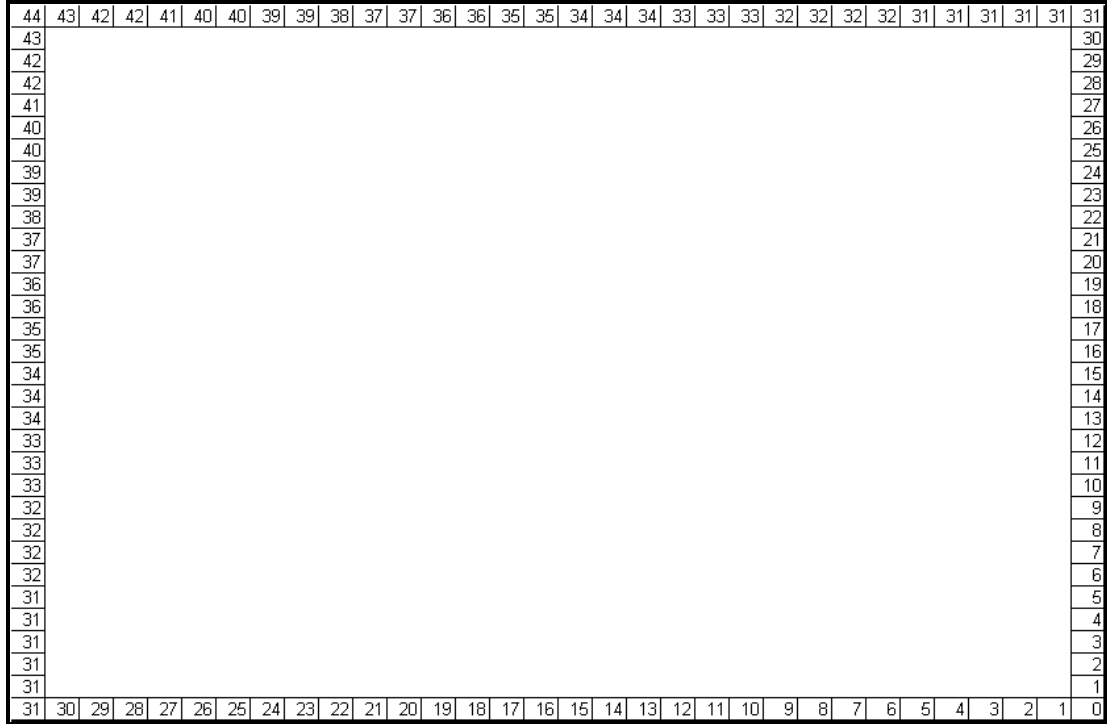


Figure 3.12: Example mobile robot map for A* algorithm with framed quadtrees
Starting position

The mobile robot starts moving towards the goal and at location A mobile robot find outs the obstacles shown in Figure 3.13. Main program calls divide function described in map building part starts to divide area till the obstacles listed in the header file are added to the robotmap. And then main program calls direction_framed function to define the direction and cost matrices. Figure 3.14 shows the direction and cost values generated by simulation program.

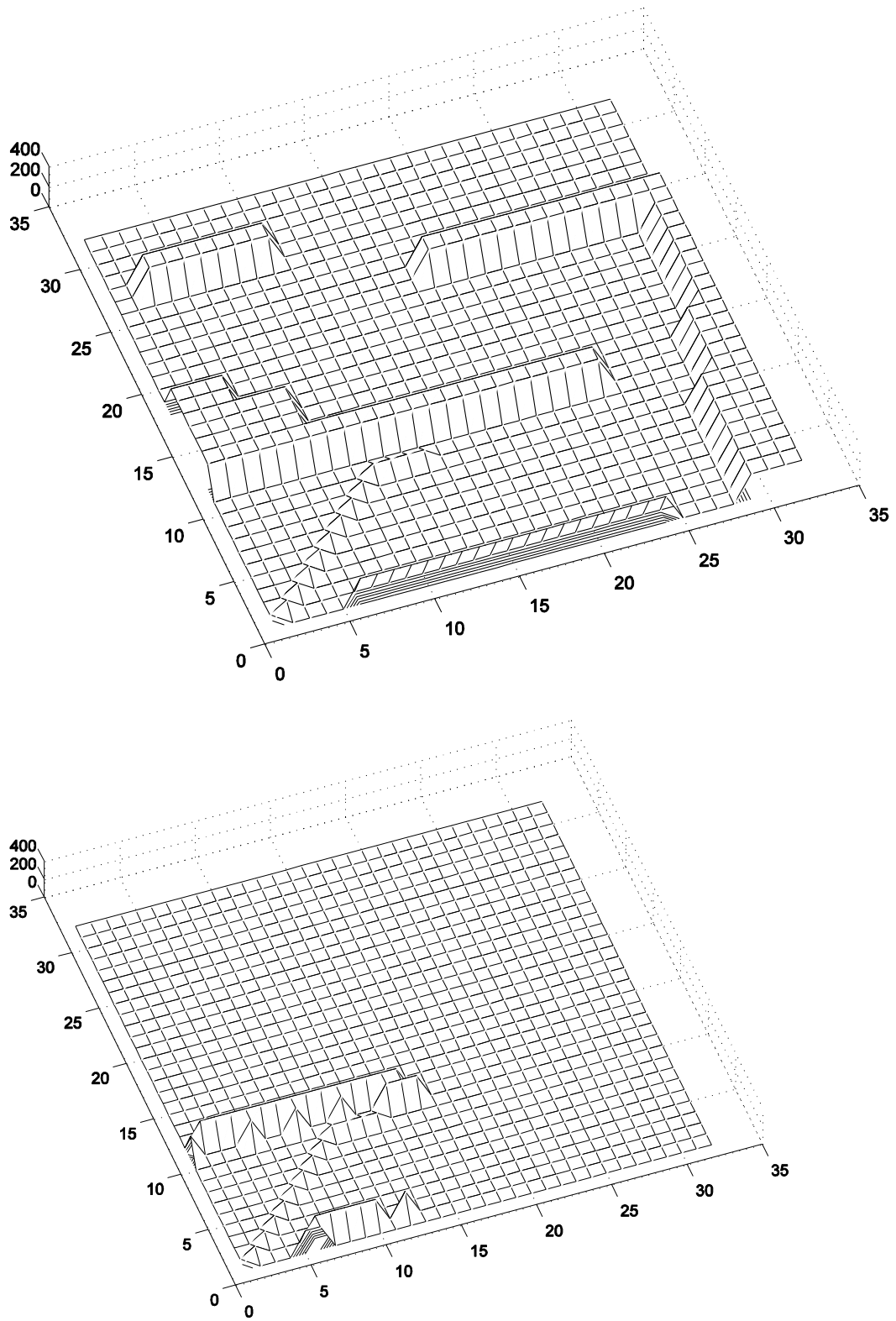


Figure 3.13: Example mobile robot map for A* algorithm with framed quadtrees
Location A

[illegible]

33

Flow chart of main program for A* with framed quadtrees given as it is shown in figure 3.15.

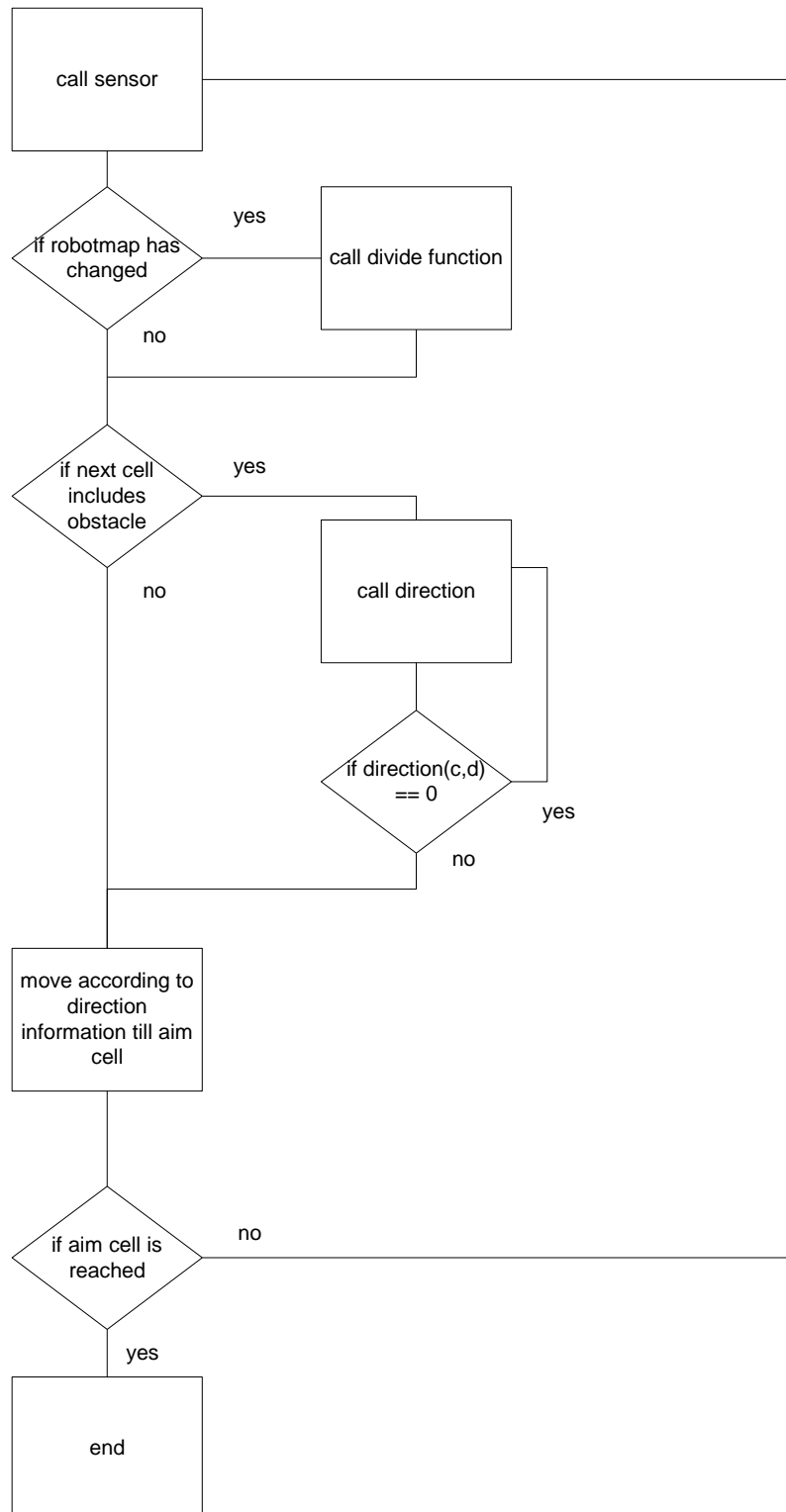


Figure 3.15: Flow chart of main program for A* with framed quad tree cells

Figure 3.16 gives flow chart of direction program for A* algorithm with framed quadtrees which defines direction, cost, openlist, values for framed quad trees.

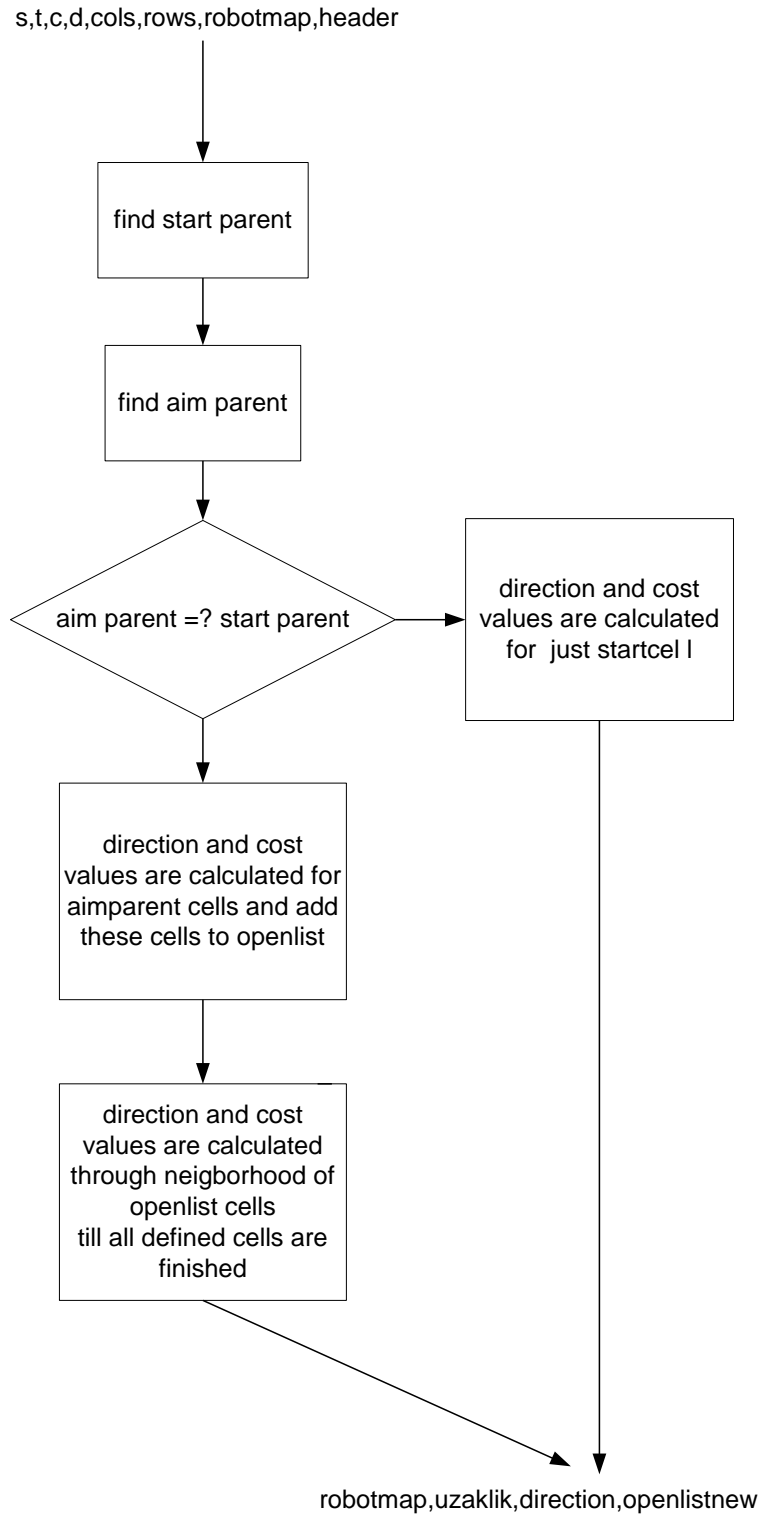


Figure 3.16: Flow chart of direction program for framed quad tree cells

Specifically, two procedures are utilized. One procedure finds quadtree neighbors whose quadtree areas intersect at a common side (known as the side-adjacent neighbors). The other procedure finds quadtree neighbors whose areas intersect at a common corner (known as the corner neighbors). After allocating a numbered list of border-cells around the perimeter of every terminal quadtree-node, border-cell neighborhood pointers are assigned. If both border-cells are in the same terminal quadtree-node, they are simply implicitly connected. On the other hand, if the border-cells belong to different terminal quadtree-nodes, it is necessary to determine if these quadtree-nodes are adjacent. If they are, we find neighboring border-cells adjacent to every border-cell. Border-cell neighborhood pointers of these border-cells are then allocated to point to each of their neighboring border-cells. In addition to these structural pointers, each border-cell has one specialized pointer, called a backpointer, for use by the D* algorithm. Figure 3.17. illustrates the typical connection patterns of a framed-quadtree (not all connections are shown for an obvious reason):

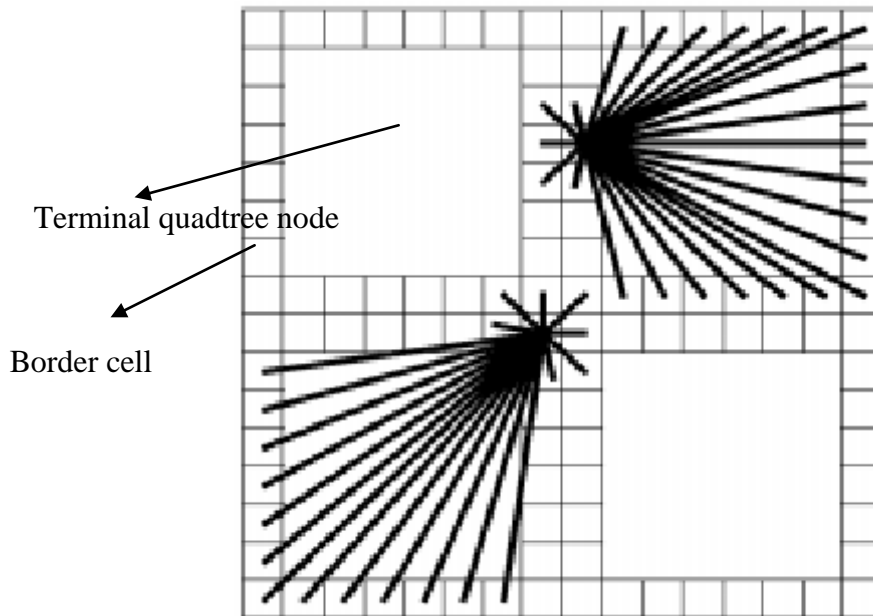


Figure. 3.17: Typical connection patterns of a framed-quadtree

Border-cells at the ends of the affected links may need their path costs recomputed. The black rectangle indicates the obstacle causing the split. The following is the procedure:

- 1-For each border-cell pair i, j in a quadtree-node that is going to be split, do:
 - Check if the neighborhood pointers between i and j are affected by the split. If they are not, then do nothing, otherwise proceed to the next step.
 - Label the link ij “to be deleted”.
 - Place both i and j on the OPEN list with their current path costs.
2. *D* will pop up cells from the OPEN list for recomputation of path costs (also known as expansion). When expanding a cell x , if a cell y has a backpointer to x through a “to be deleted” link, place y back on the OPEN list with its path cost set equal to the maximum cost MAXCOST and set y 's backpointer to NULL.*
3. *Delete all links connected to cell x that are to be deleted.*

In using framed quadrees to propagate the path planning wave, several problems arise in determining the next step in the movement. First of all to obtain framed-quadtree structure, algorithm should be divided by taking care to known obstacles robots position and the aim. Next step is to determine the moving steps direction and cost matrices between the cells by direction algorithm for framed quad trees.

After obtaining the framed quadtree structure by dividing, then all the cells link to others according to the costs. Direction function starts with the aim cell and then all cells in the same parent with aim cell send to open list .

After the goal quad has been reached, a fine path through specific cells and quads may be found by simply tracing the path back from the goal to the robot through the ‘origin’ pointers. This is the shortest path from the robot to the goal, based on framed quadtree algorithm.

3.2. The D* Algorithm

The problem with A* approach is strictly computational: replanning is an expensive operation. If the robot's prior map information is sparse or inaccurate, then too much time will be spent replanning in Step 2 of A* algorithm for the approach to be viable. The D* algorithm [35] was developed to solve this computational problem.

Robotmap changes to the arc costs are likely to be in the vicinity of the robot, since it typically carries a sensor with a limited range. This means that most plans need only be patched "locally". Also, the robot generally makes near-monotonic progress toward the goal. Because of these reasons there is no need to path planning starting with the goal each time. Instead of this just the local robots circumference can be updated by D* algorithm. Thus, D* enables replanning to be computationally viable even when the map information does not match the environment very well.

Like A*, D* uses a graph of states to represent the robot's environment, where each state is a robot configuration and the arcs adjoining the states represent the cost of travelling from state to state. Initially, D* computes the cost of reaching the goal to every state in the graph given all known cost information. As the robot follows a sequence of states to the goal, it may discover a discrepancy between the map and the environment. All paths routed through this arc are invalidated and must be "repaired" [3,29,35,36].

The connections or arcs are labelled with positive scalar values indicating the cost of moving between the two cells. Each cell (also called a "state") includes an estimate of the path cost to the goal, and a backpointer. The backpointer of each border-cell points to the neighboring border-cell having the least cost to the goal. Following backpointers from any cell, the optimal path from the cell is recovered. Initially, the goal state is placed on the OPEN list with an initial cost of zero. The state with the minimum path cost on the OPEN list is repeatedly expanded, propagating path cost calculations to its neighbors. States on the list are processed in order of increasing key value. The process can terminate when the lowest value on the OPEN list equals or exceeds the robot's path cost, since additional expansions cannot possibly find a better path to the goal. Once a new optimal path is computed or the old one is determined to be valid, the robot can continue to move toward the goal. The robot then begins to move, following the backpointers toward the goal. While driving, the

robot scans the terrain with its sensor. If it detects an obstacle where one is not expected, then all optimal paths containing this arc are no longer valid. D* updates the robotmap and places the adjoining state on the OPEN list, then repeatedly expands states on the OPEN list to propagate the path cost increase along the invalid paths. The OPEN list states that transmit the cost increase are called RAISE states. As the RAISE states fan out, they come in contact with neighbor states that are able to lower their path costs. These LOWER states are placed on the OPEN list. Through repeated state expansion, the LOWER states reduce path costs and redirect backpointers to compute new optimal paths to the invalidated states. Conversely, if the robot's sensor detects the absence of an obstacle where one is expected, then a new optimal path may exist from the robot to the goal through the "missing" obstacle. D* updates the robotmap, places the adjoining state on the OPEN list as a LOWER state, then repeatedly expands states to compute new optimal paths wherever possible. In either case, D* determines how far the cost propagation must proceed until a new optimal path is computed to the robot or it is decided that the old one is still optimal. Once this determination has been made, the robot is free to continue moving optimally to the goal, scanning the terrain for obstacles.

If we consider the same example that we have used in A* algorithm it is easy to see the difference of the algorithms. When the mobile robot reaches location A and detects the obstacles needs to update its direction and cost matrices. To do this D* algorithm looks for which cells calculations can be keep same. Figure 3.18 shows the mobile robot direction and cost matrices kept same.

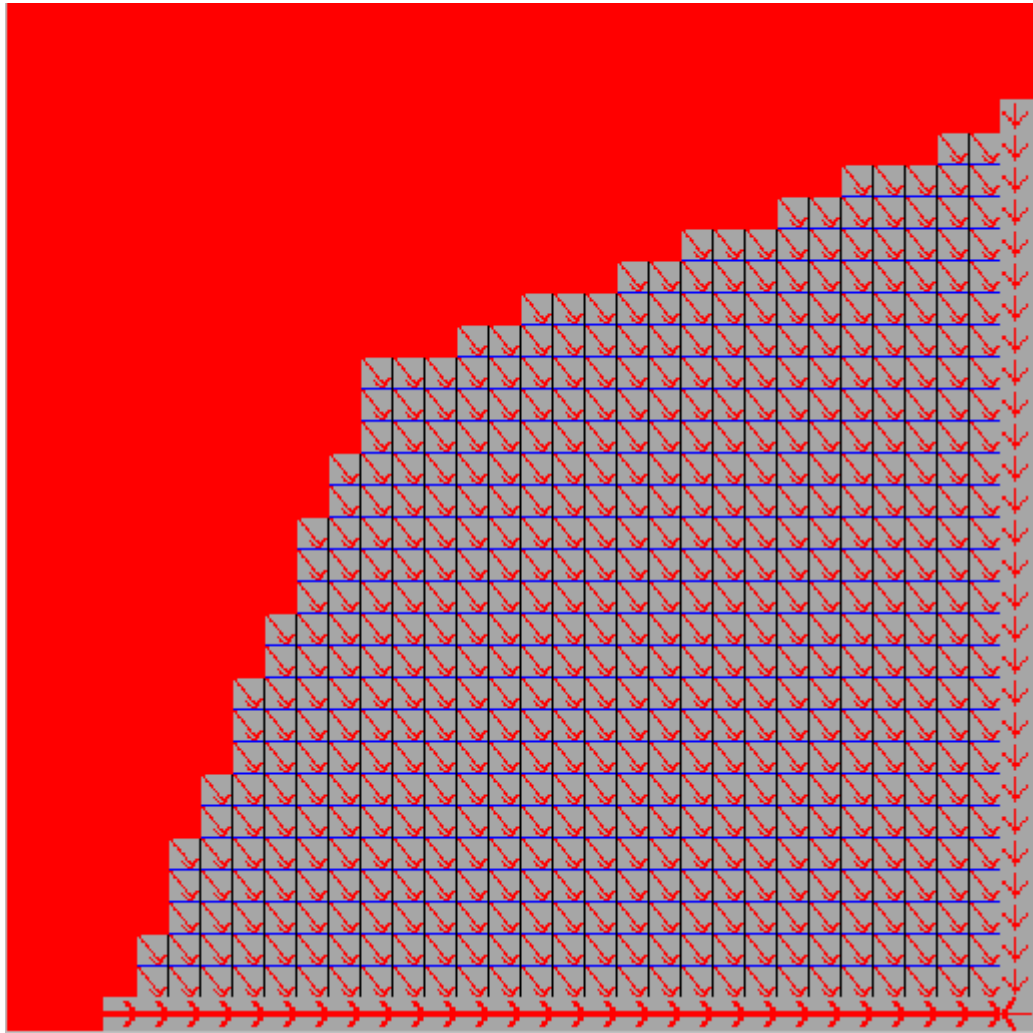


Figure 3.18: Direction and cost matrices cells which are kept same at location A for D* algorithm

Then the cells which have the max cost values listed in open list and the search algorithm starts. Figure 3.19 shows the last situation of the direction and cost matrices.

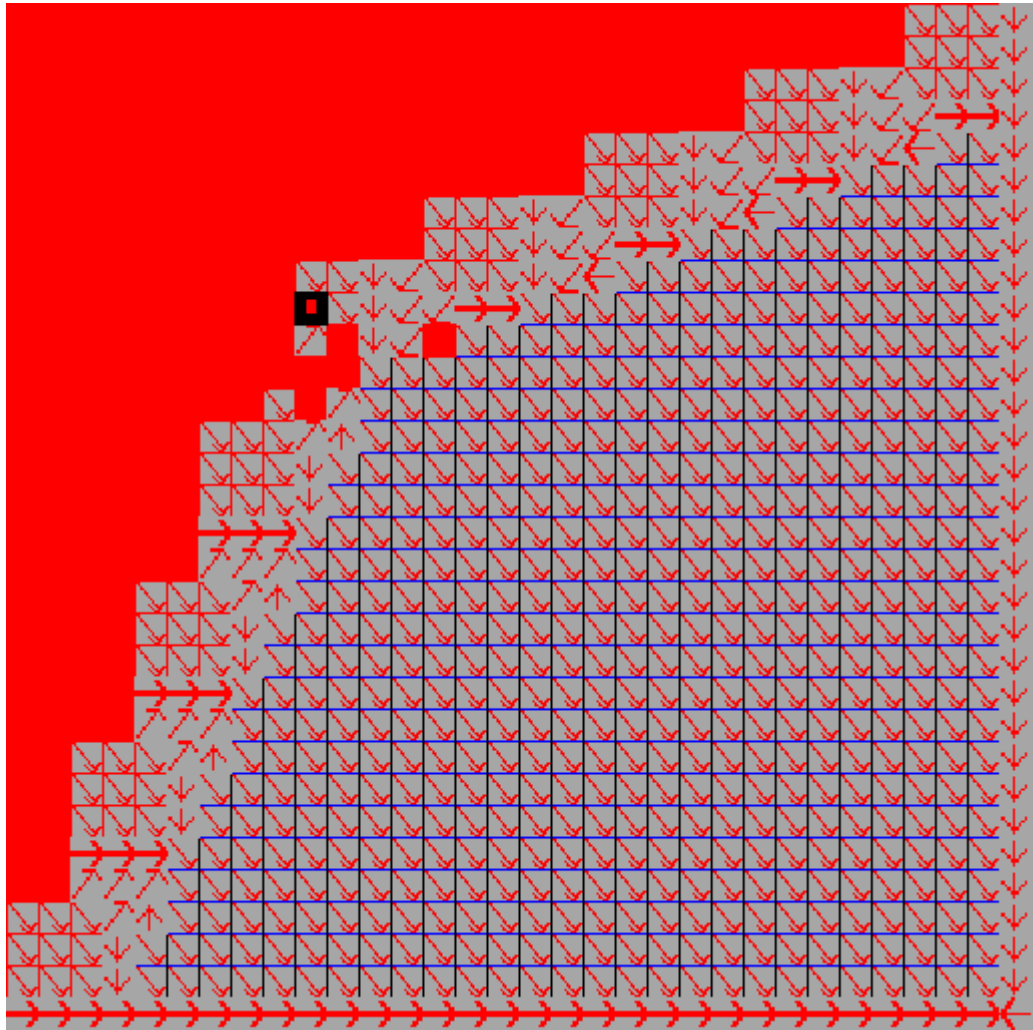


Figure 3.19: Direction and cost matrices updated at location A for D* algorithm

Here the difference between A* and D* algorithms are shown by graphics. It is clear that D* algorithm needs to update much less cell values than A* algorithm and this makes it faster.

4. SENSOR MODEL

To create the environment map, the robot system must interpret the signals it receives from its sensors and obtain an understanding of its surroundings by acquiring a model of its environment and workspace. The information of the physical features of an environment are taken from different positions along the path followed by the robot by external sensors. Once the world model is obtained, it can be used to improve the quality of the paths, to locate a target, to help object recognition or to define expectations in the trajectory of the mobile robot. Another benefit of the developing of map building for mobile robots is they will be aware of the changes around them as people do. Furthermore, from the commercial point of view, operators prefer a mobile robot, which can immediately be put into operation without the need to create or obtain a map of the robot's new environment beforehand.

To make the use of robots feasible in real-life applications, it is necessary to reach a tradeoff between costs and benefits. Often, this prevents the use of expensive sensors (e.g., video cameras) in favor of cheaper sensing devices, and calls for efficient algorithms that can guarantee real-time performance in the presence of insufficient or conflicting data. Typically, on-board sensors for a given robot are chosen according to several criteria: view angle, range capability, accuracy and resolution, real-time operation, redundancy, simplicity, size and power consumption. Popular sensors typically include sonar, tactile, infrared and laser sensors and videocameras. The selection basically depends on what kind of behaviour is expected from the robot. Navigational behaviours often rely on sonar sensors despite their obvious drawbacks. Even though sonar sensors have a wide arc of uncertainty and only provides information about the distance to the closest obstacle in the beam direction, they are light, cheap, fast, easy to process and have a long detection range. The advantages of using sonar sensors for navigation are widely discussed in [6, 18].

During the perception phase, ultrasonic sensors are fired in such a sequence that interference phenomena are minimized, and measures are recorded together with the position of the corresponding sensor. A single reading provides the information that

one or more obstacles are located somewhere along the 25° arc of circumference of radius r . Hence, while points located in the proximity of this arc are likely to be occupied, there is evidence that points well inside the circular sector of radius r are empty.

Ultrasonic range finders measure the distance from obstacles in the environment by a simple conversion of the time of flight of the ultrasonic waves in air. These are constituted by a single transducer acting both as a transmitter and a receiver; a packet of ultrasonic waves is generated and the resulting echo is detected. The time delay between transmission and reception is assumed to be proportional to the distance of the sensed obstacle.

The problem of building a map from ultrasonic measures is made difficult by the large amount of uncertainty introduced by the sensing process. This uncertainty consists in a lack of evidence: due to the inherent limitations of ultrasonic sensors, it is not always possible to decide whether a given point of the area of interest is occupied or not by an obstacle. Rather than classifying points of the space as either empty or occupied in this unfavorable situation, a possible alternative approach is to convey all the available knowledge into an uncertain representation. In fact, an ultrasonic sensor detects the closest reflecting surface inside its radiation cone, thereby indicating the presence of an empty space up to a certain distance. On the other hand, no information is provided about the state of the area beyond such distance: the available evidence does not suggest emptiness or occupancy. Only by incorporating measures taken at different viewpoints it will be possible to discriminate between the two possibilities.

A very simple probability distribution is used to model the cells in a sonar scan. The occupancy probability of a cell is modelled as:

$$P(\theta, \rho) = \begin{cases} -\varepsilon, & \text{if } 0 \leq \rho \leq d - \delta, 0 \leq |\theta| \leq \beta/2 \\ +\varphi, & \text{if } d - \delta \leq \rho \leq d + \delta, 0 \leq |\theta| \leq \beta/2 \\ 0, & \text{otherwise} \end{cases} \quad (5.1)$$

ρ and θ being the distance to the robot and the angle of the main axis of the sonar beam, respectively; d the range measurement returned by the sonar sensor and β the beam aperture. 2δ determines the width of the region of uncertainty where the obstacle could be located. Finally, the empty and occupied probability density

functions for a cell inside the sonar beam are given by constants ϵ and ϕ respectively. The current model of the sonar sensor is shown in Figure 4.1. [18].

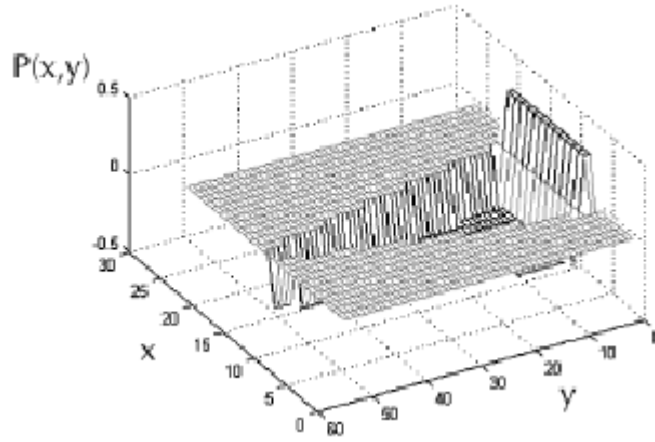


Figure. 4.1: Current model of the sonar sensor.

The multilobed beam pattern of the transmitter can be obtained from the radiation directivity function of a plane circular piston

$$D(v) = 2 \frac{J_1(\omega p \sin v)}{\omega p \sin v} \quad (5.2)$$

where $J_1(.)$ is the first-order Bessel function, $\omega = 2\pi/l$ depends on the wavelength l , p is the piston radius, and v is the azimuthal angle measured with respect to the beam central axis. For the Polaroid sensor [17], it is $p = 0.01921$ m and $l = c/v$, where c is the sound speed in air and $v=49.410$ kHz. For practical purposes, it is sufficient to take into account only the principal lobe of the pattern. As a consequence, the waves are considered to be diffused over a radiation cone of 25° width [37].

A single range reading is affected by three basic sources of uncertainty as follows.

- The sensor has a limited radial resolution. The standard Polaroid range finder can detect distances from 0.12 to 6.5 m with 1% accuracy over the entire range.
- The angular position of the object that originated the echo inside the radiation cone is not determined. For example, all the three obstacles of Figure 4.2 will give the same distance reading.

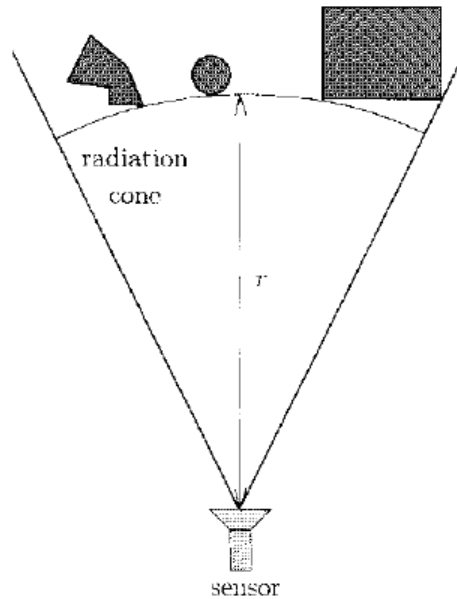


Figure 4.2: Objects in different positions can give the same distance reading by ultrasonic sensing

- Specular reflection is a phenomenon where the energy being emitted by the sensor strikes an adjacent object but the incidence angle is sufficiently shallow that it causes the return echo to be reflected away from the detector. Even though the obstacle is close, the sensor returns a maximum range reading erroneously indicating the space is open. If the incidence angle is larger than a critical value Φ , the sensor reading is not significant because the beam may reach the receiver after multiple reflections, or even get lost (see Figure 4.3.). The angle Φ depends on the surface characteristics, ranging from 7° to 8° for smooth glass to almost 90° for very rough materials. Specular reflection has the adverse effect of sweeping out large areas of the map beyond adjacent obstacles. However, a similar case arises when there is no obstacle within range of the sensor. In this case, no obstacle is present so no echo is returned. Consequently, the sensor times out and returns a value indicating maximum range. The sensor model is adjusted to update only the areas inside the arc and ignores its edges. Given the limited sensor range of most sensors and the potentially vast spaces of the environment, maximum range readings account for a large portion of the makeup of the map [5].

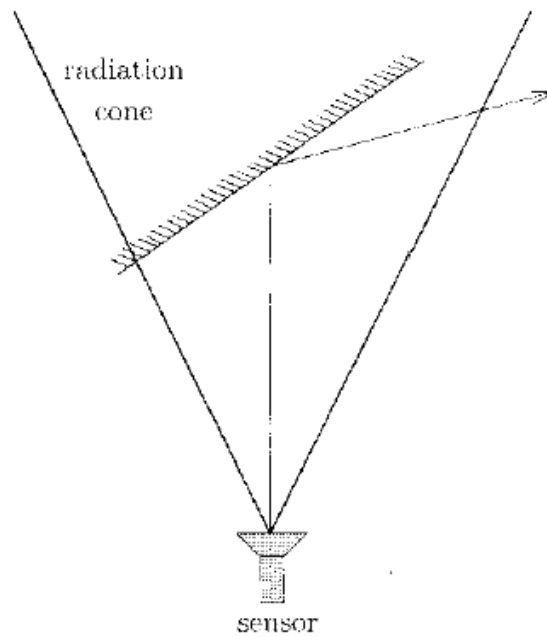


Figure 4.3: Specular reflection: False reflections may occur for large angles of incidence by ultrasonic sensing

The sensor function flow chart for the ultrasonic sensor simulation is given below with Figure 4.4.

Sensor function receives robotmap, realmap, current robot position, and angle, map size and current obstacle list from the main program. Sensor number is the number of sensors located on mobile robot. It is assumed that these sensors are arranged equally on front side of mobile robot in 180° .

Sensor distance is the max distance that sensor can read any information. According to the above information algorithm looks for obstacles in the realmap if there is no obstacle than do nothing but if there is an obstacle which is not find before than update robotmap , update obstacle list and set mapchange flag so direction program should run.

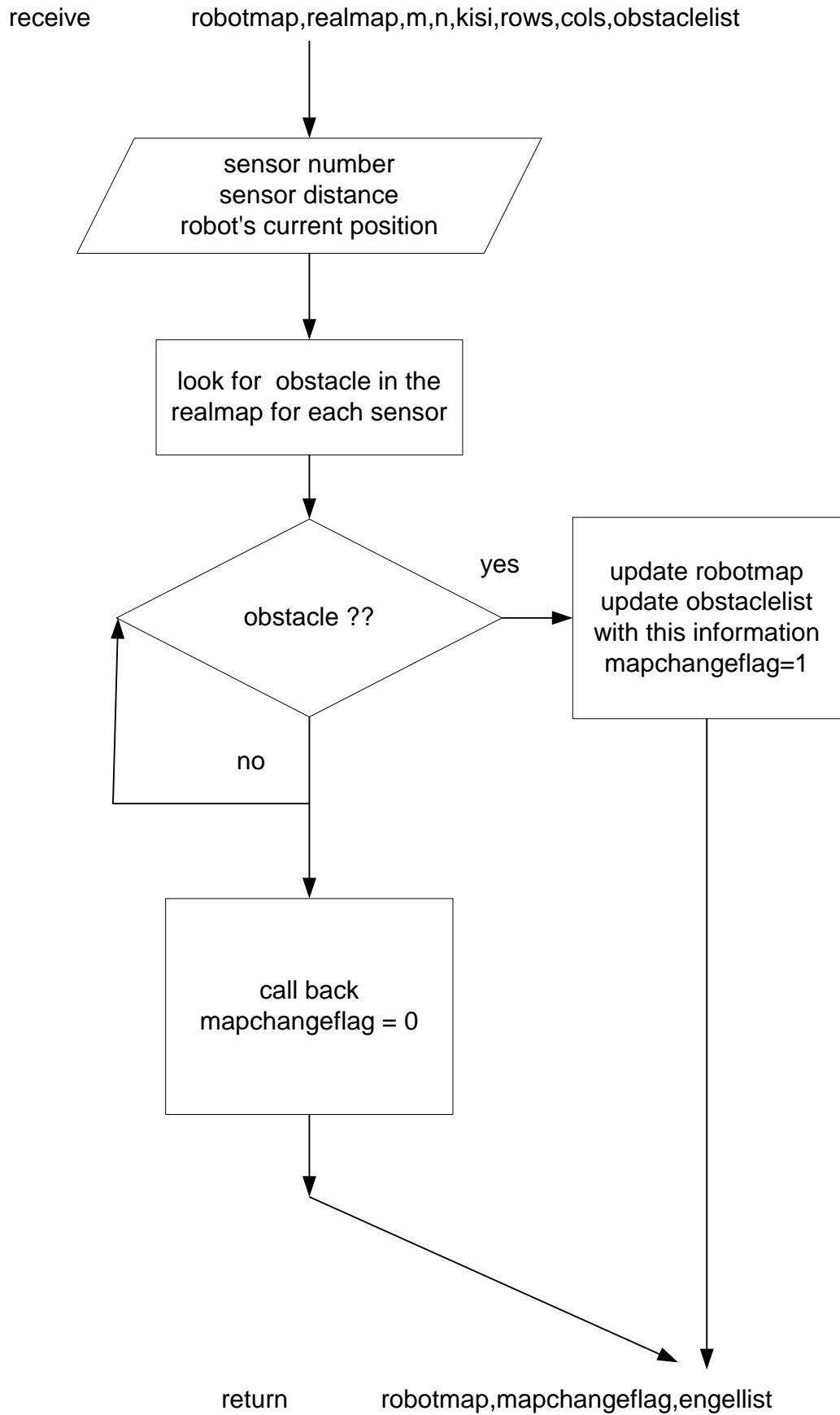


Figure 4.4: Sensor function flow chart

Sensor simulation is shown below by a simple example . Figure 4.5 gives the realmap and robot is located at 19,28 point. In this case the robot's sensors are fired and the result is given as robotmap with Figure 4.6. 6 obstacles are detected by ultrasonic sensors.

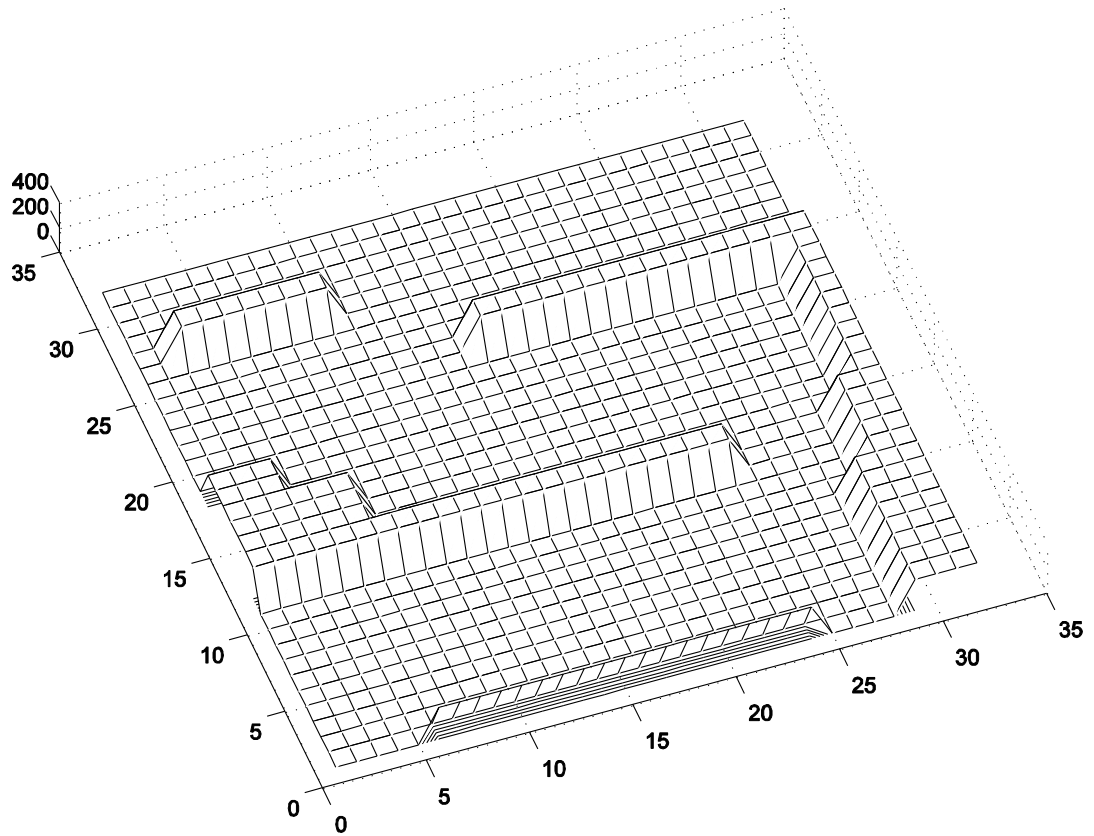


Figure 4.5: Realmap for sensor simulation

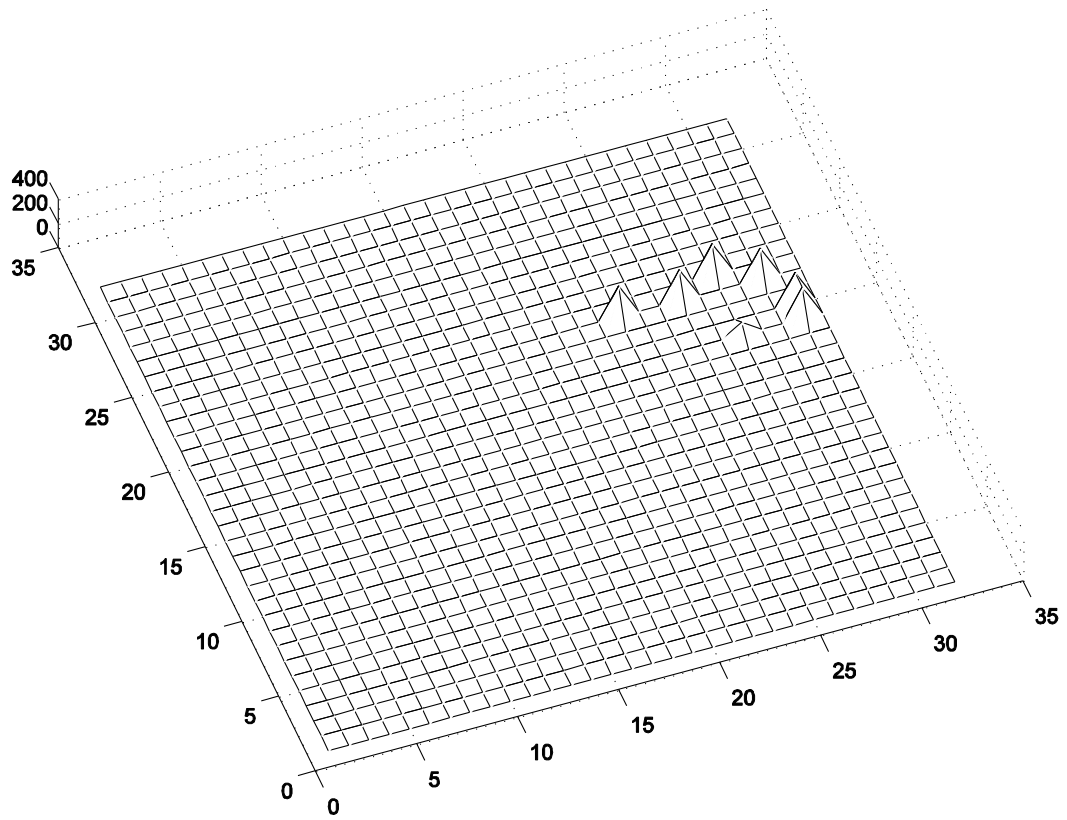


Figure 4.6: Robotmap after ultrasonic sensors has fired.

In this example

Sensor number is 5

Sensor distance is 5

$K_{si} = \pi$

First sensor's angle is 0

5. SIMULATION STUDIES

We have run an extensive set of simulations using Matlab program to compare the performance of D* and A* path planning algorithms when used with framed quadtrees as opposed to regular grids in incrementally discovered environments. The simulation environments are shown in figure 5.1, figure 5.2 and figure 5.3 with different cell sizes (32*32, 64*64, 128*128,..) and with the same obstacle configuration. The robot is located at (1,1) position of the map and the goal is located at the other corner which is (32,32) or (64,64) or any other corner.

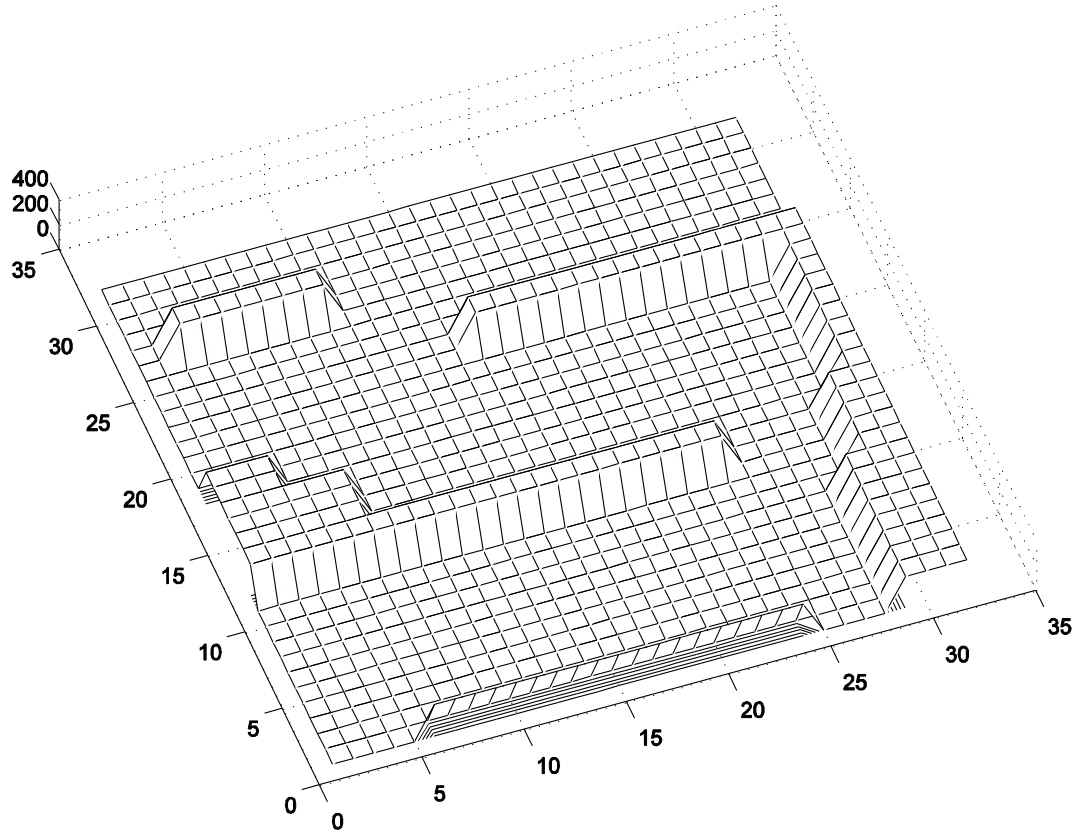


Figure 5.1: Realmap for simulation (32*32)

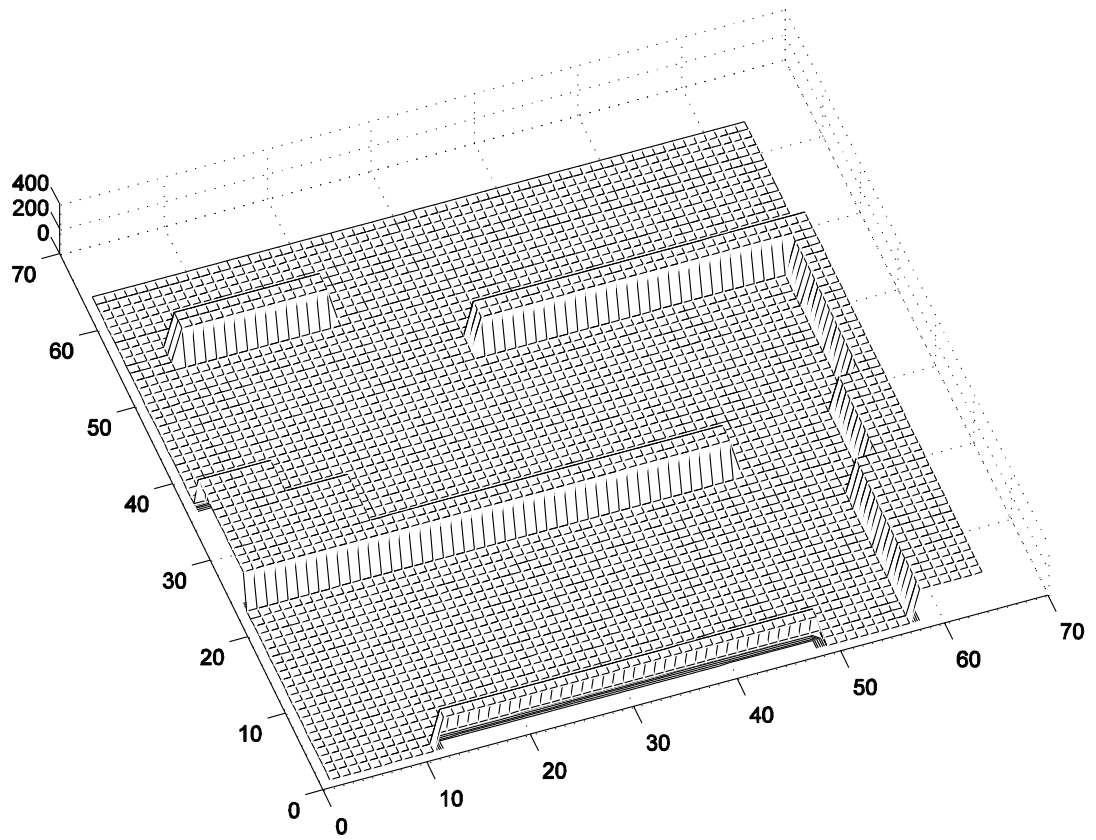


Figure 5.2: Realmap for simulation (64*64)

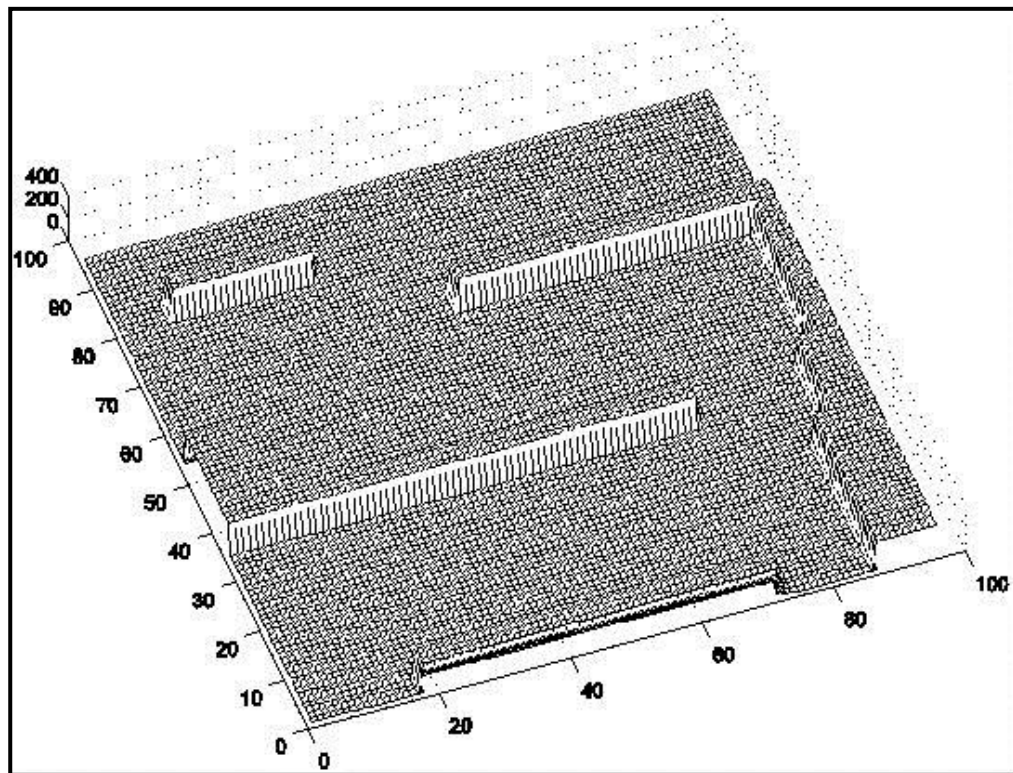


Figure 5.3: Realmap for simulation (96*96)

And continues with 128*128 ,160*160 cell size environments with the same obstacle positions.

If the terrain is passable, the cost to move from one cell to another is the Euclidean distance and if the terrain is impassable the cost to move to a cell containing an obstacle is infinite. The environment is completely unknown so, all obstacles must be discovered by mobile robot's sensors. In the simulations it is assumed that the mobile robot is able to detect obstacles with 5 sensors located on mobile robot as shown in figure 5.4. It is assumed that each sensor can detect any obstacle which is located in 5 cell distance in 25° beam aperture.

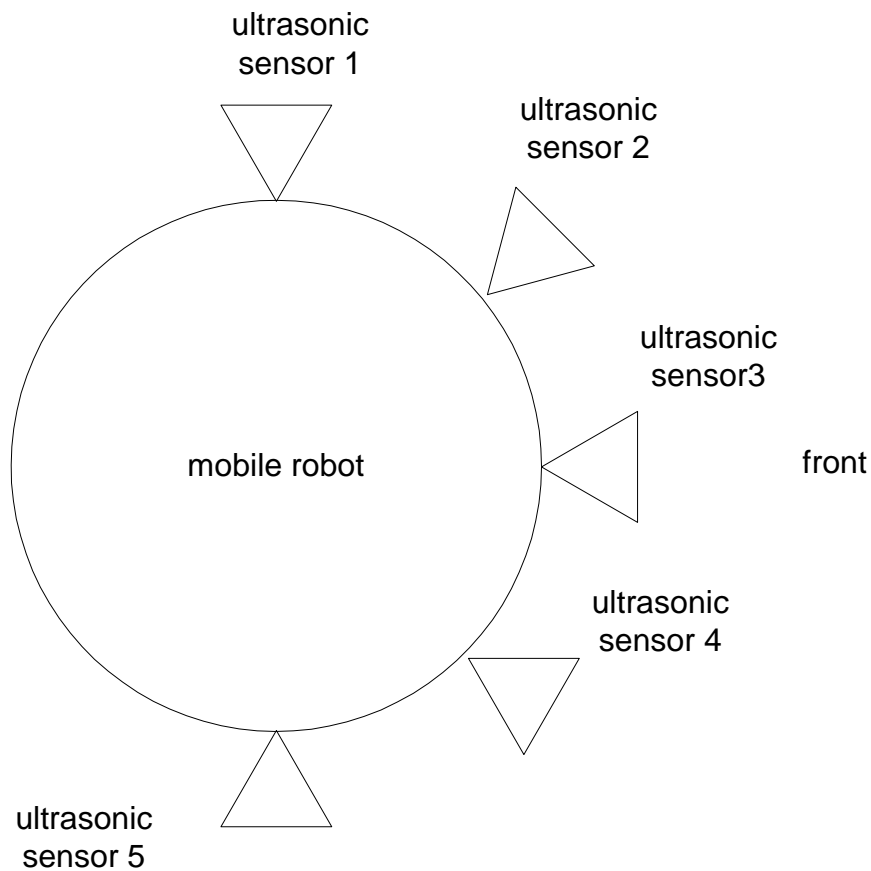


Figure 5.4: ultrasonic sensor positions on mobile robot

Below we compare simulation results of regular grids, quadtrees and framed quadtrees with A* and D* path planning algorithms using three criteria: traverse length, memory usage, and execution time.

5.1. Traverse Length

Traverse length is measured in cell units. Horizontal and vertical traverses between smallest cells count as 1 unit, while diagonal traverses count as $\sqrt{2}$ units. Traverses through a large empty area (that is, across a large framed-quadtree cell) is calculated with euclidean rules.

As it is seen on the paths traverses generated using framed quadtrees are shorter than those generated when regular grids are used primarily because the path is not forced to travel on diagonals. This effect is particularly noticeable when the environment is sparse. Correspondingly, the difference in traverse length is smaller as the cell size become smaller.

Figure 5.5 shows the comparison of traversel lenth between regular grid and framed-quadtree mapping.

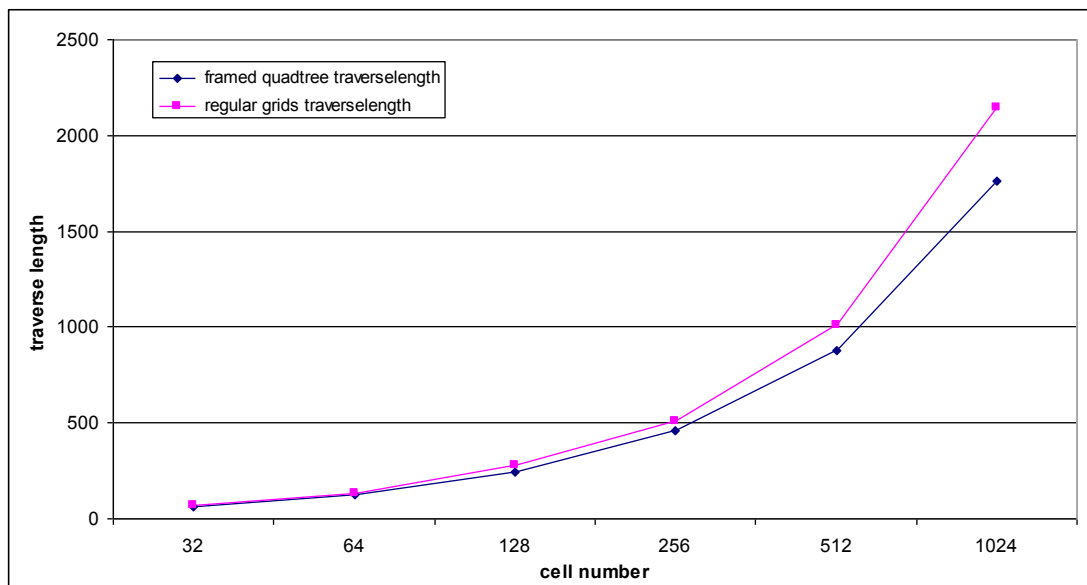


Figure 5.5: Comparison of traverse length between regular grid and framed-quadtree mapping

5.2. Memory Usage

Memory usage is the maximum memory (in bytes) used by the program during a run, matlab program gives this information by using 'whos' command. Memory usage changes with path planning algorithm and map building techniques , in Figure 5.6. used memory results are summarized. The results for A* and D* path planning algorithms with framed-quadtree and regular grid map building algorithms. Figure

5.6 shows that framed quadrees use less memory in the range of cell sizes with which we have experimented. Our simulation results show that for unknown worlds, framed quadrees reduces memory usage by over 40%.

When path planning algorithms A* and D* (used with regular grids) compared it is seen from the simulation results that D* uses less memory than A* . When map building algorithms regular grid and framed-quadtree (used with A* path planning) framed-quadtree mapping uses less memory. It is clear that D* algorithm with framed quadtree mapping uses mininum memory.

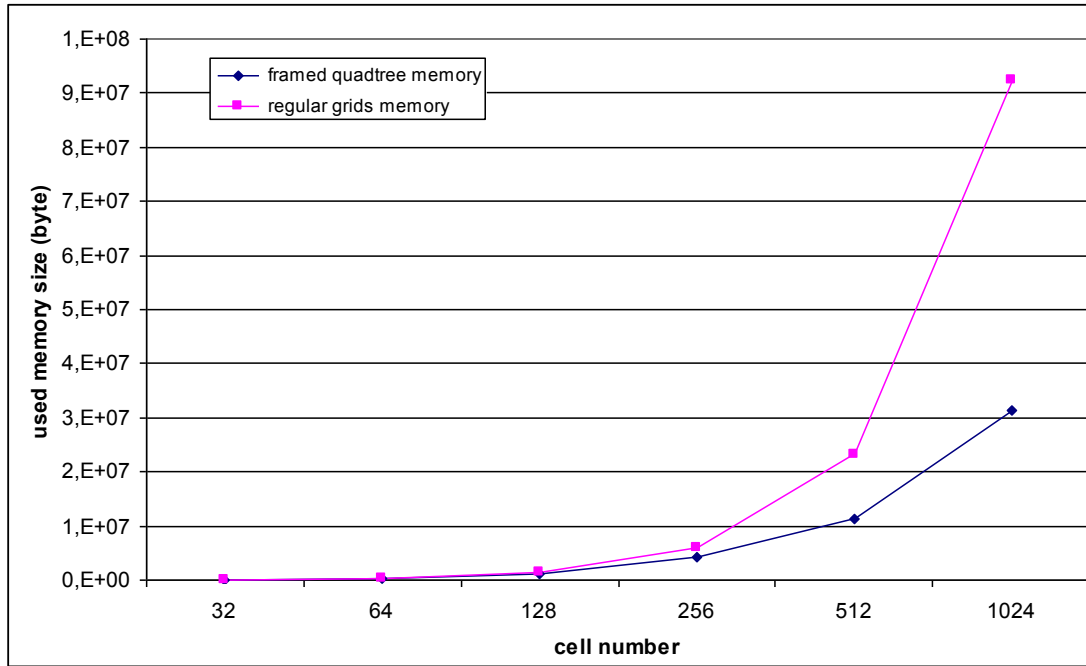


Figure 5.6: Comparison of memory usage in between regular grid and framed-quadtree mapping

5.3. Execution Time

We have compared the execution time of A* and D* path planning algorithms when a regular grid or framed-quad tree mapping is used. Results are summarized in Figure 5.7. Total time is measured in seconds. For a completely unknown environment, the total time is consistently lower when framed quadrees are used. The use of framed quadrees significantly reduces the time over regular grids, since very few cells are needed to represent the assumed free space.

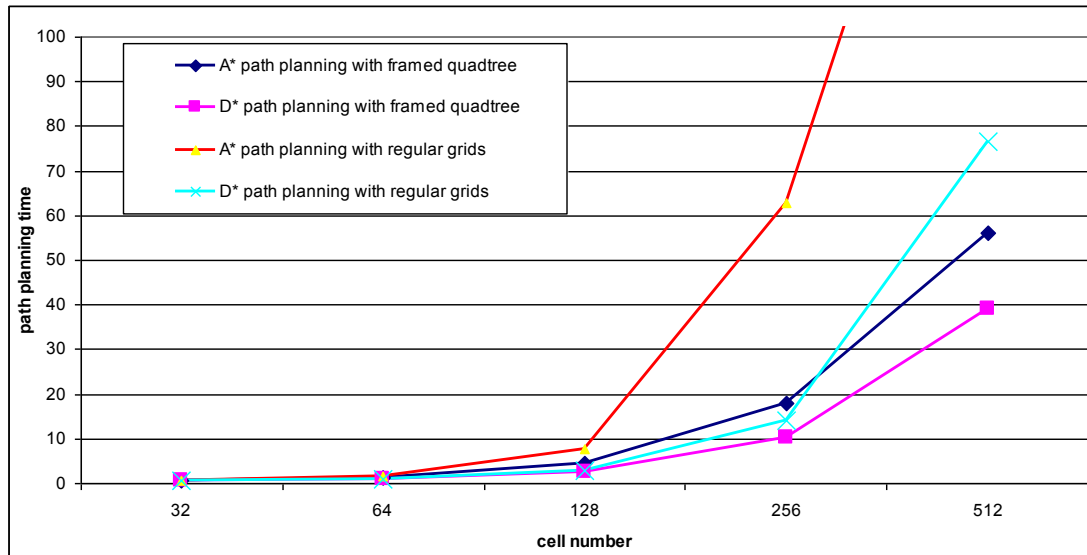


Figure 5.7 : Execution time comparison

Table 5.1 summarizes the simulation results as a table. It is easy to find the best mapping method and planning technique for this study by using this table.

Table 5.1: Summary of results

Summary of results		Mapping method		
		Regular Grids	Quadtree Grids	Framed-Quadtree Grids
planning technique	A*	Travel Length ☹️ Memory Usage ☹️ Execution Time ☹️	Travel Length ☹️ Memory Usage ☹️ Execution Time ☹️	Travel Length 😊 Memory Usage ☹️ Execution Time ☹️
	D*	Travel Length ☹️ Memory Usage ☹️ Execution Time 😊	Travel Length ☹️ Memory Usage 😊 Execution Time 😊	Travel Length 😊 Memory Usage 😊 Execution Time 😊

By using this simulation program it is possible to analyse any environment to choose the best mapping method and path planning algorithm. For the environment map given in figure 3.1 according to the table 5.1 framed-quadtree mapping should be used with D* path planning algorithm

6. CONCLUSION

Outdoor navigation of mobile robots equipped with ultrasonic sensors are analysed for map building and path planning algorithms. To do this a simulation environment is created by using Matlab program. A sample environment map is created and for different cell sizes the results of simulations are compared. Traverse length, time and memory usage are the main criteria for comparison of regular grid and framed-quad tree mapping techniques, A*, D* path planning algorithms. Simulation results shows how the use of framed-quadtrees leads to paths that are shorter and more direct than when other representations like regular grids and quadtrees are used. Combining an optimal path planning algorithm like D* with framed-quadtree map representation has the benefit of optimal path planning in traverselenth and time while minimizing the memory requirements.

In this study the simulation results couldn't be tried on an actual mobile robot and mobile robot movement characteristics couldn't modeled. In the future work this study can be applied on actual mobile robot with mobile robot movement characteristics. Also this study will be integrated with SICK laser sensor and some sensor fusion algorithms can be added.

REFERENCES

- [1] **Schraft R. D. and Volz H.**, 1996. *Serviceroboter, Innovative Technik in Dienstleistung und Versorgung*, Springer-Verlag, Berlin, Heidelberg.
- [2] **Dudek, G., and Jenkins, M.**, 2000. *Computational Principles of Mobile Robotics*. Cambridge University Press, Cambridge, USA.
- [3] **Wettergreen D., et. al.**, 2003. "Life in the Atacama: Field Season 2003 Experiment Plans and Technical Results," Carnegie Mellon University Robotics Institute Technical Report, CMU-RI-TR-03-50.
- [4] **Tompkins P., Stentz A., Whittaker W.**, 2003. "Field Experiments in Mission-Level Path Execution and Re-Planning", The Robotics Institute, Carnegie Mellon University 5000 Fobes Avenue, Pittsburgh, PA 15213.
- [5] **Oriolo G., Ulivi G., Vendittelli M.**, 1998. Real-Time Map Building and Navigation for Autnomous Robots in Unknown Environments. *IEEE Transactions on systems, man, and cybernetics* Vol.28, No: 3.
- [6] **Urdiales C., Bandera A., Perez E., Poncela A., Sadoval F.**, 2002. Hierarchical Planning in a Mobile Robot for Map Learning and Navigation. Universidad de Malaga, 29071 Malaga-Spain.
- [7] **Lee, D.**, 1996. *The Map-Building and Exploration Strategies of Simple Sonar-Equipped Mobile Robot*, Cambridge Univ. Press., Cambridge.
- [8] **Elfes, A.**, 1987. Sonar-based real-world mapping and navigation, *IEEE J. Robotics Automat.* **3**, 249–265.
- [9] **Zelinsky, A.** 1991. Mobile Robot map making using sonar, *J. Robotics Systems*, 557–577.
- [10] **Gonzalez, J., Ollero, A., and Reina,** 1994. A Map building for a mobile robot equipped with 2D laser rangefinder, in: *Proc. of the 1994 IEEE Internat. Conf. on Robotics and Automation*, **3**, 1904–1909.
- [11] **Gonzalez, J., Stentz, A., and Ollero, A.**, 1992. An iconic position estimator for a 2D laser rangefinder, in: *Proc. of the 1992 IEEE Internat. Conf. on Robotics and Automation*, **3**, 2646–2651.
- [12] **Gasos, J. and Martin, A.**, 1996. A fuzzy approach to build sonar maps for mobile robots, *Computers in Industry* **32**, 151–167.
- [13] **Gasos, J. and Rosetti, A.**, 1999. Uncertainty representation for mobile robots: Preception, modeling and navigation in unknown environments, *Fuzzy Sets Systems* **107** , 1–24.

- [14] **IP Y. L., RAD A. B., CHOW K. M. and WONG Y. K.,** 2002. Segment based map building using enhanced adaptive fuzzy clustering algorithm for mobile robot applications. *Journal of Intelligent and Robotic Systems* **35**: 221–245.
- [15] **Kambhampati S. and David L.S.** 1985. Multiresolution path planning for mobile robot,in: *IEEE Journal of Robotics and Automation*,Vol. RA-2, No.3, 135-145.
- [16] **Zelinsky, A.,** 1992. “A Mobile Robot Exploration Algorithm”. *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 6, 707-717.
- [17] **Yahja A., Singth S. and Stentz A.,** 2000. Efficient On-Line Path Planner for Outdoor Mobile Robots, *Robotics and Autonomous Systems*, **32(2)**, 129-143.
- [18] **Leonard, J.J. and Durrant-Whyte, H.F.,** 1992. Directed Sonar Sensing for Mobile Robot Navigation. Kluwer Academic Publishers, Massachussets.
- [19] **Stentz A., Hebert M.,** 1995. A Complete Navigation System for Goal Acquisition in Unknown Environments. The robotics Institute Carnegie Mellon University, Pittsburgh, Pennsylvania.
- [20] **Goto, Y., Stentz, A.,** 1987. “Mobile Robot Navigation: The CMU System,” *IEEE Expert*, Vol. 2, No. 4.
- [21] **Lumelsky, V. J., Stepanov, A. A.,** 1986. “Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment”,*IEEE Transactions on Automatic Control*, Vol. 31, No. 11.
- [22] **Pirzadeh, A., Snyder, W.,** 1990.“A Unified Solution to Coverage and Search in Explored and Unexplored Terrains Using Indirect Control”, *Proc. of the IEEE International Conference on Robotics and Automation*, May.
- [23] **Korf, R. E.,** 1987. “Real-Time Heuristic Search: First Results,” *Proc. Sixth National Conference on Artificial Intelligence*, July.
- [24] **Nilsson N.J.,** 1980. Principles of Artifical Intelligence. *Tioga Publishing Company*, pp.72-88.
- [25] **Boult T.,** 1987. Updating distance maps when objects move. *In Proceedings of the SPIE Conference on Mobile Robots*.
- [26] **Trovato K.I.,** 1990. Differential A*:an adaptive search method illustrated with robot path panning for moving obstacles and goals,and an uncertain environment. *Journal of Pattern Recognition and Artifical Intelligence*,**4(2)**.
- [27] **Ramalingam G. and Reps T.,** 1992. An incremental algorithm for a generalization of the shortest-path problem.University of Wisconsin Technical Report#1087.
- [28] **Stentz A.,** 1994. Optimal and efficient path planning for partially-known environments. *In Proceedings of the IEEE International Conference on Robotics and Automation*.

- [29] **Stentz A.**, Map-Based Strategies for Robot Navigation in Unknown Environments, Carnegie Mellon University, Pittsburgh, Pennsylvania U.S.A.
- [30] **Samet H.**, 1998. An overview of quadtrees, octrees, and related hierarchical data structures, in: NATO ASI Series, Vol. F40, 51-68.
- [31] **Chen D.Z., et. al.**, 1995. Planning conditional shortest paths through an unknown environment: a framed-quadtree approach, in: Proc. 1995 IEEE/RJS *International Conference on Intelligent Robots and Systems, IROS*, **3**, 33-38.
- [32] **Kelly A.**, 1995. An intelligent predictive control approach to the high speed cross country autonomous navigation problem, *Ph.D. Thesis*, Robotics Institute, Carnegie Mellon University.
- [33] **Stentz A., Singh S., Yahja A., Brumitt B. L.**, 1998. Framed-Quadtree Path Planning for Mobile Robots Operating in Sparse Environments. *In proceedings, IEEE conference on Robotics and Automation Belgium*, May.
- [34] **Russell S. and Norvig P.**, 1995. Artificial Intelligence: A Modern Approach, Prentice Hall, Singapore.
- [35] **Stentz A.**, 1995. The Focussed D* algorithm for real time replanning. *In proceedings of the International Joint Conference on Artificial Intelligence*, August.
- [36] **Ramalingam G. and Reps T.**, 1992. An incremental algorithm for a generalization of the shortest-path problem. *University of Wisconsin Technical Report 1087*.
- [37] **Bandera A., Urdiales C., Sandoval F.**, 2000. An hierarchical approach to grid-based and topological maps integration for autonomous indoor navigation. Dpto. Tecnologia Electronica, E.T.S.I. Telecomunicacion Universidad de Malaga, Spain.

CURRICULUM VITAE

Murat Gündoğdu was born in Bolu, Turkey, in 1979. He was graduated from Bolu İzzet Baysal Anatolian High School in 1997. He received B.Sc. degree in mechanical engineering in 2002 and he received B.Sc. degree in Electronics and communication engineering in 2003, both from the Istanbul Technical University. He started his M. Sc. program in mechatronics in 2002 . He joined Arçelik in 2003 and since then he works as design engineer in the product development department of Arçelik.